TRƯỜNG ĐẠI HỌC KỸ THUẬT TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN PHẦN MỀM HỆ THỐNG
---o0o---

LUẬN ÁN TỐT NGHIỆP

Đề tài:

XÂY DỰNG CÔNG CỤ HỖ TRỢ CHO XỬ LÝ SONG SONG TRÊN HỆ THỐNG PHÂN BỐ

Thầy giáo hướng dẫn : TS. NGUYỄN THANH SƠN
Sinh viên thực hiện : ĐẶNG TRẦN KHÁNH
TRƯƠNG HỒNG LĨNH
Lớp : MT93

12/1997

🕮 NHẬN XÉT CỦA THẦY HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

🕮 NHẬN XÉT CỦA THẦY PHẢN BIỆN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

LỜI NÓI ĐẦU

Ngày nay máy tính là một công cụ không thể thiếu được của con người . Từ những nhân viên làm việc ở văn phòng cho đến các kỹ sư với những bài toán có hàng triệu phép toán , các nhà khoa học đang làm việc trong các phòng thí nghiệm , ... đều cần có sự hỗ trợ tích cực của máy tính .

Tuy nhiên với một nền khoa học kỹ thuật ngày càng phát triển đương nhiên sẽ kéo theo sự xuất hiện của nhiều dự án lớn trong nhiều lĩnh vực khác nhau . Trong số đó có nhữmg dự án mà muốn hoàn thành thì cần phải giải các bài toán với khối lượng tính toán rất lớn . Cũng vẫn trong số các bài toàn đó , có những bài toán mà kết quả của nó chỉ có ý nghĩa nếu được hoàn thành trong một khoảng thời gian cho phép nào đó như các bài toán tính toán trong hệ thống thời gian thực (real time sytem) , các bài toán tính quĩ đạo chuyển động của các tên lửa , các vệ tinh , các bài toàn tính toán nhằm phục vụ cho dự báo thời tiết , các bài toán về xử lý ảnh , trí tuệ nhân tạo , ...

Để giải được những bài toán cần khối lượng tính toán lớn với những ràng buộc về thời gian như vậy thì bên cạnh việc cải tiến công nghệ , tăng tốc độ xử lý của máy tính người ta còn chia bài toán ra thành những bài toán (công việc) nhỏ hơn và được xử lý song song trên nhiều processor khác nhau . Lý thuyết toán học và thực tế đã chứng minh rằng thời gian tính toán của các bài toán đó sẽ được giảm xuống đáng kể nếu như chúng ta phân chia bài toán và sử dụng các processor một cách hợp lý .

Ngày nay đã có những bài toán với nhiều processor (CPU) rất tiện lợi cho việc giải các bài toán như trên , tuy nhiên giá của chúng rất cao và chính vì thế nên lợi ích mà chúng đem lại vẫn chưa có sức thuyết phục lớn đối với những người (user) sử dụng máy tính bình thường. Một giải pháp tối ưu hơn là sử dụng mạng máy tính sẵn có (mỗi node trên mạng chỉ cần là các máy tính có một processor) , bài toán của chúng ta được nhiều máy tính trên mạng cùng xử lý để cho ra kết quả giống như chúng ta mong muốn . Tuy nhiên để hiện thực được ý tưởng đó thì chúng ta cần có một môi trường để thực thi . Môi trường thực thi ở đây có thể là bất cứ một mạng máy tính nào với hệ điều hành đa nhiệm (multitasking) được cài đặt trên mạng . Do yêu cầu của đề tài nên môi trường thực thi được chọn ở đây là môi trường mạng Unix .

Mạng phân bố Unix có những đặc điểm và tính chất thích hợp cho phép chúng ta thực hiện ý tưởng "dùng mạng phân bố để xử lý song song" này . Thật vậy , Unix là một hệ điều hành đa nhiệm , có khả năng cho phép nhiều quá trình thực thi đồng thời (có thể trên một hoặc nhiều host khác nhau) . Chúng ta sẽ phân chia bài toán lớn thành các bài toán nhỏ hơn một cách hợp lý và mỗi bài toàn nhỏ đó sẽ được giải quyết trên các máy (gọi là các host) trong mạng (ở đây là mạng Unix) của chúng ta . Mặt khác các bài toán nhỏ này là một phần của bài toán lớn nên chúng có mối quan hệ với nhau . Vì vậy khi một bài toán nhỏ được chạy trên một host như một quá trình của hệ điều hành Unix thì nó có thể có nhu cầu cần trao đổi thông tin với các quá trình khác trên các host khác của mạng (chúng là các đại diện để xử lý một phần khác của bài toán lớn - một bài toán nhỏ khác) . Với hệ điều hành Unix chúng ta có thể làm được điều này : hệ điều hành Unix cung cấp cho chúng ta một vài phương thức để truyền dữ liệu giữa các quá trình trong một host hoặc giữa các host khác nhau trong mạng .

Mặc dù hệ điều hành Unix cung cấp cho chúng ta những phương tiện hết sức thuận lợi để thực thi ý tưởng " dùng mạng phân bố để xử lý song song " nhưng quản lý được một chương trình gồm có nhiều module và mỗi module lại được phân bổ lên các host trên mạng (mỗi

module đại diện cho một bài toán nhỏ) không phải là một việc dễ dàng : người lập trình cần có một kiến thức vững vàng về hệ thống , phải tổ chức , lưu trữ các cấu trúc dữ liệu và thực hiện trao đổi thông tin giữa các module , ... Công việc này vừa không dễ dàng mà lại tốn khá nhiều thời gian (phải lặp lại công việc mỗi khi có bài toán mới được tiến hành theo mô hình này) .

Xuất phát từ lý do đó và để tạo điều kiện cho người lập trình có thời gian tập trung vào những ý tưởng , những thuật giải mới , giảm bớt công sức và thời gian nghiên cứu hệ thống , ... chúng ta cần thiết phải tạo ra một công cụ (tool) để hỗ trợ cho người lập trình khi giải những bài toán liên quan tới mô hình xử lý song song và phân bố trên mạng . Đây cũng chính là ý tưởng cũng như là yêu cầu của đề tài và đã được hai sinh viên khóa 93 thực hiện là Đặng Trần Khánh và Trương Hồng Lĩnh .
Bản thuyết minh luận án này gồm có 6 phần chính :

- Phần 1 : Giới thiệu hướng thực thi đề tài .
- Phần 2 : Giao diện đồ họa với người sử dụng .
- Phần 3 : Thiết kế và thực thi phần hệ thống của đề tài .
- Phần 4 : Thư viện cho người lập trình .
- Phần 5 : Tổng kết công việc và hướng phát triển của đề tài .
- Phần 6 : Chương trình nguồn .

Trong phần 1 , độc giả sẽ biết khái quát về những công việc mà các tác giả đã thực hiện . Ở phần 2 , các độc giả sẽ được hướng dẫn một cách chi tiết về cách sử dụng phần mềm và cũng sẽ biết thêm các thông tin , các nguyên tắc khác khi sử dụng phần mềm . Tiếp theo phần 3 sẽ giới thiệu chi tiết các công việc ở mức hệ thống mà các tác giả đã làm . Nếu độc giả chỉ quan tâm tới cách sử dụng phần mềm và không quan tâm tới phần thiết kế mức hệ thống của đề tài thì có thể bỏ qua phần thuyết minh này của đề tài , tuy nhiên độc giả vẫn phải đọc phần 4 để biết được các hàm thư viện của phần mềm nhằm tạo được cầu nối để thực hiện sự trao đổi dữ liệu giữa các module (là các bài toán nhỏ hơn) được chạy trên một host hoặc các host khác nhau của mạng Unix . Phần 5 sẽ tổng kết lại các công việc mà hai sinh viên đã làm được cũng như các hướng mở để phát triển đề tài . Phần cuối cùng là chương trình nguồn dành cho các độc giả tham khảo nếu muốn biết chi tiết hơn nữa về quá trình xây dựng đề tài của đồng tác giả Đặng Trần Khánh và Trương Hồng Lĩnh .

Vì nhiều điều kiện khách quan cũng như do trình độ của các tác giả nên phần mềm còn có những hạn chế . Rất mong được có sự đóng góp ý kiến từ các độc giả . Các ý kiến đóng góp xin được gởi về hoặc liên hệ trực tiếp với địa chỉ :

Đặng Trần Khánh (MT93) Email : dtkhanh@dit.hcmut.ac.vn
Trương Hồng Lĩnh (MT93) Email : thlinh@dit.hcmut.ac.vn
Khoa Công Nghệ Thông Tin Trường Đaih Học Kỹ Thuật TP Hồ Chí Minh
hoặc qua điện thoại của khoa Công Nghệ Thông Tin : (08) 8658689 .

Cuối cùng chúng em xin chân thành cảm ơn các thầy cô đã tận tình dìu dắt chúng em trong suốt quá trình học tập cũng như khi thực hiện đồ án tốt nghiệp này . Đặc biệt chúng em xin gửi lời cảm ơn tới thầy Sơn , thầy Nam , thầy Hoài , thầy Đạt và thầy Hiến đã giúp đỡ về

tài liệu cũng như đóng góp những ý kiến cho chúng em trong quá trình làm luận án tốt nghiệp.

Một lần nữa xin cảm ơn gia đình và bè bạn đã nhiệt tình giúp đỡ trong qúa trình học tập và làm đồ án tốt nghiệp .

12 / 1997

Mục Lục

Nhận xét của thầy hướng dẫn.... 1
Nhận xét của thầy phản biện.... 2
Lời nói đầu.... 3
Mục lục.... 6

Phần 1 : Giới thiệu hướng thực thi đề tài 9

Phần 2 : Giao diện đồ họa với người sử dụng.... 11

1. Làm quen với giao diện đồ họa của DPPT.... 11
2. Chức năng của các Menu,các Button và cụ thể về các vùng trong giao diện với .... user của DPPT.... 13
   2.1. Vùng Network System Area 13
   2.2. Vùng User's Application Area.... 14
   2.3. Chức năng của các Button - Vùng Quick Tool.... 14
   2.4. Chức năng của các Menu trong thanh Menu ngang.... 33
   2.5. Tóm tắt qui trình thao tác với DPPT của các User 45

Phần 3 : Thiết kế và thực thi phần hệ thống của đề tài 47

Chương 1 : Thiết kế và thực thi phần hệ thống của giao diện với người sủ dụng 48

1. Giới thiệu chung.... 48
2. Quản lý các host trong hệ thống của DPPT.... 48
   2.1. Cấu trúc dữ liệu của các host.... 49
   2.2. Kỹ thuật tìm kiếm các host trong mạng 51
   2.3. Các vấn đề liên quan khác.... 55
3. Quản lý các node trong hệ thống của DPPT.... 55
   3.1. Các cấu trúc dữ liệu 56
   3.2. Các vấn đề liên quan 60
4. Kỹ thuật Compile một node của DPPT.... 60
5. Kỹ thuật Run một node của DPPT 64

Chương 2 : Phân tích , thiết kế các chương trình hệ thống và thư viện lập trình 66

A. Phân tích các yêu cầu 67

1.Vấn đề trao đổi dữ liệu giữa các process trên hệ thống 67
   1.1Mô hình trao đổi dư liệu dùng chung 67
   1.2 Interprocess Communication 72
   1.3 Data representation 73
2. Vấn đề quản lý hệ thống 74

2.1 Quản lý các đối tượng trong hệ thống 74
2.2 Fault tolerance and recovery 74
2.3. Dead lock 74

B. Các công cụ lập trình được hệ thống hỗ trợ 76

1. Multithreaded Programming 76
1.1 Programming with thread 78
1.2 Programming with Synchronization Object 79
2. Interprocess comunication 79
2.1 Pipes 80
2.2 System V IPC 80
3. Networking Interface 82
3.1 Communication protocols 82
3.2 Sockets 82
4. External Data Representation (XDR) 83
4.1 XDR Data Types 83
4.2 XDR Library Routines 83

C. Thiết kế và hiện thực chương trình 85

1.Thiết kế và hiện thực phần hệ thống của tool 85
1.1 Phần hệ thống cho Port Server dùng quản lý thông tin 85
1.2 Phần hệ thống cho Network Shared Memory Server (NSM Server) 90
1.3 Phần hệ thống cho Message Passing 104
1.4 Các chương trình dùng setup và quản lý hệ thống 113
1.5 Faul tolerance and Recovery 118
2. Các hàm chuyển đổi data 119
2.1 Sơ đồ tổng quát quá trình chuyển đổi data 119
2.2 Quá trình chuyển đổi dữ liệu trong tool 120
2.3.Các vấn đề còn lại 121
3. Các thành phần khác 121

Phần 4 :Thư viện cho người lập trình 122

1. Setup tool và các lệnh hệ thống của tool 123
1.1 Setup tool 123
1.2 Các lệnh hệ thống của tool 123
2. Các hàm cung cấp cho user task 124
2.1 Các hàm thao tác trên NDSM Server 124
2.2 Các hàm trao đổi message thông qua MsgPassDaemon 130
2.3 Các hàm trao đổi message trực tiếp 131
2.4 Hàm chuyển đổi dữ liệu sang dạng chuẩn 132

Phần 5 : Tổng kết công việc và hướng phát triển của đề tài 135

1. Các công việc đã hoàn tất 135
2. Hướng phát triển của đề tài 136

Tài liệu tham khảo 139
Chương trình nguồn 141

PHẦN 1

GIỚI THIỆU HƯỚNG THỰC THI ĐỀ TÀI

Ý tưởng của đề tài thực ra là "dùng mạng phân bố để xử lý song song" do đó các tác giả đã thống nhất đặt tên cho phần mềm của mình là DPPT (Distributed and Parallel Programming Tool).

Ý tưởng trên thực ra cũng đã được một sinh viên khóa 90 và sau đó là hai sinh viên kỹ sư II khóa 4 (Đại Học Bách khoa và Đại Học Kỹ Thuật) hiện thực trên môi trường HP Unix và môi trường Solaris System V . Trên thế giới thì ý tưởng này cũng được hiện thực với nhiều phần mềm khá nổi tiếng như PVM (Parallel Virtual Machine) , như Paralex , CODE , Pocker , LGDF , Alex , Fel , apE , ... và mỗi phần mềm đều có điểm mạnh và điểm yếu riêng của chúng . Việc so sánh và đánh giá giữa DPPT và các phần mềm trên chúng tôi xin dành cho độc giả sau khi đọc xong báo cáo này và chạy thử DPPT với những bài toán trong thực tế của các độc giả (!?) .

DPPT được sử dụng trên mạng Unix nhưng không nhất thiết các máy trên mạng phải là đồng nhất (homogeneous) mà CPU của các host có thể thuộc các loại khác nhau . Đối với user (người sử dụng) thì công việc chuyển dổi dữ liệu giữa các hệ máy khác nhau là rất dễ dàng vì công việc này đã được DPPT cung cấp bẳng các hàm thư viện . Trong thực tế , khi làm luận án tốt nghiệp thì các tác giả đã thử sử dụng DPPT đồng thời trên CPU SPARC của SUN và trên các CPU họ INTEL của các máy PC và chương trình đều chạy tốt .

Một điểm mạnh nữa của DPPT là nhiều user (multiuser) có thể sử dụng phần mềm DPPT một cách an toàn không hề có khả năng đụng độ về phần hệ thống bên dưới cũng như tên các file source code (bao gồm include files) của user. Đặc điểm này sẽ được giới thiệu kỹ ở phần 2.

DPPT được thiết kế và xây dựng gồm hai phần chính :

- *Phần giao tiếp với người sử dụng gồm :*
  - Phần giao diện đồ họa với người sử dụng (Graphics User Interface - GUI) : cho phép user phân chia bài toán lớn thành các bài toán nhỏ hơn đồng thời thực hiện các chức năng generate , map , compile hay run các module , delete các module ,... hoàn toàn bằng chuột (mouse). Các chức năng này và qui trình thao tác trên GUI của DPPT sẽ được giới thiệu một cách chi tiết ở phần 2 .

- Phần các hàm thư viện dành cho người sử dụng để thực hiện trao đổi dữ liệu giữa các module chạy trên một host hoặc trên các host khác nhau trong mạng . User không cần quan tâm đến mức thấp hơn của hệ thống như cách truyền nhận data , cách quản lý các port , ...

- *Phần hệ thống mức thấp của DPPT như :* các cấu trúc dữ liệu để quản lý các module (mỗi module trong DPPT được biểu diễn là một node của đồ thị dạng cây) , cấu trúc dữ liệu quản lý các host , kỹ thuật tìm kiếm các host , quản lý các port tham gia vào bài toán hiện thời , ... Phần này là hoàn toàn ẩn (hide) đối với các end user tạo cho họ sự thoải mái khi làm việc với công cụ này và đây cũng là mục đích chính của việc xây dựng DPPT : các user chỉ tập trung vào bài toán của họ , phát triển các ý tưởng , ... mà không cần quan tâm đến mức hệ thống của mạng .

Việc thực thi ý tưởng thiết kế đó được thực hiện bằng ngôn ngữ C++ trong *môi trường X - Window* của hệ điều hành Unix với sự trợ giúp của phần mềm tạo giao diện Sparcword/Visual của hệ thống SOLARIS .

Yêu cầu đối với user sử dụng DPPT là phải biết lập trình bằng ngôn ngữ C , C++ tuy nhiên với ***bất cứ*** ngôn ngữ lập trình nào có cấu trúc của compiler dạng :

<COMPILER> <SOURCE FILE> -o <DESTINATION FILE> [LIBS]

thì đều dùng được trong DPPT .

Bên cạnh những điểm mạnh của DPPT thì do nhiều điều kiện cả khách quan lẫn chủ quan cho nên vẫn còn nhiều điểm hạn chế khi làm việc với DPPT . Những điều này sẽ được chỉ ra một cách cụ thể và đề nghị hướng để cải tiến , sửa đổi ở phần 5 . Bây giờ chúng ta sẽ đi vào phần giới thiệu về cách sử dụng GUI của DPPT cũng như các thông tin chi tiết khi tạo ra các chức năng trong GUI .

PHẦN 2

GIAO DIỆN ĐỒ HỌA VỚI NGƯỜI SỬ DỤNG

---

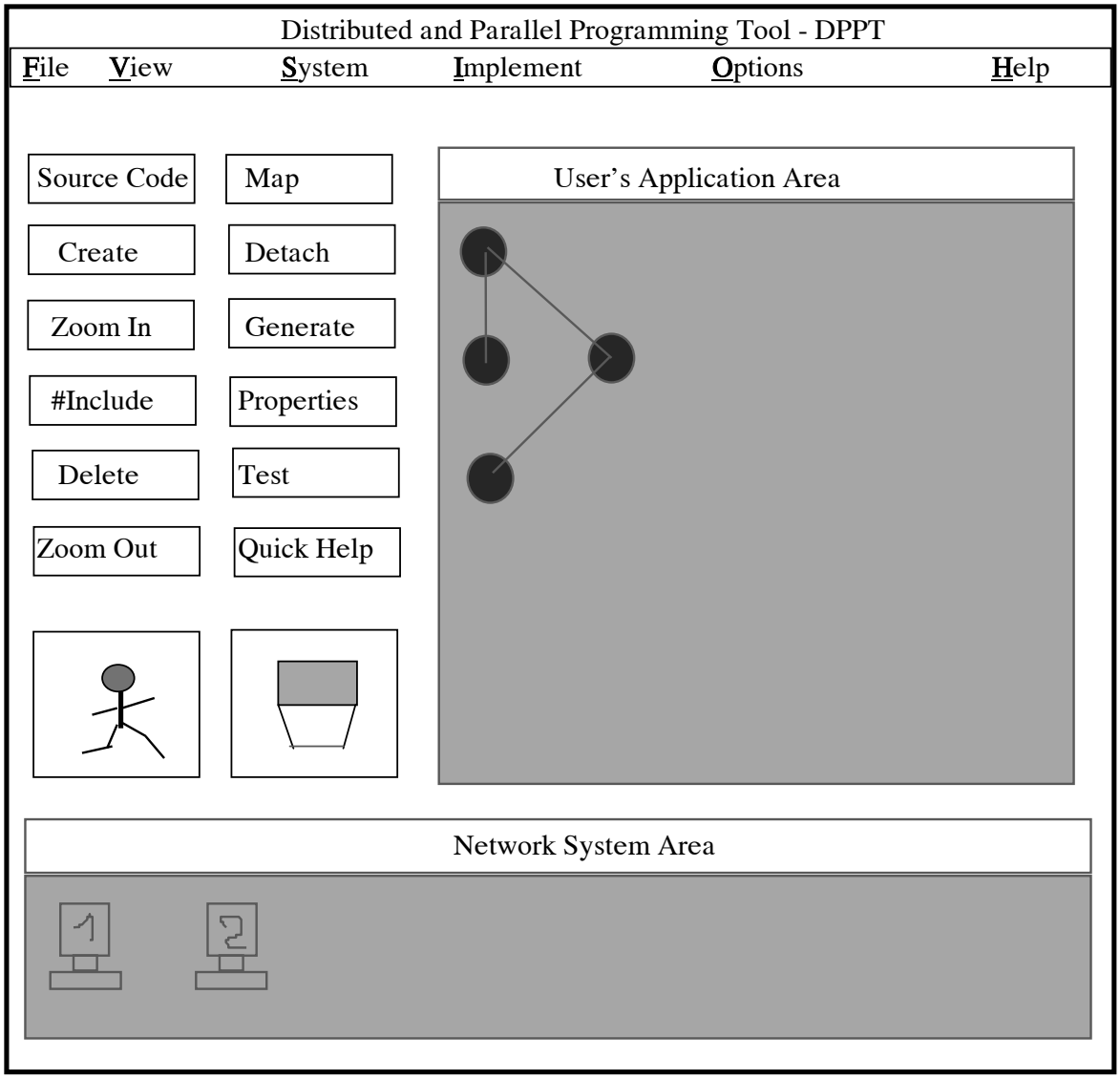
1. Làm quen với giao diện đồ họa của DPPT

Trên hệ thống mạng của chúng ta có nhiều host và một bài toán lớn (module hay node 0) được chia ra làm nhiều bài toán nhỏ hơn là các module (node) con và cho các module con chạy đồng thời trên những host khác nhau . Để giải quyết được điều này màn hình giao diện đồ họa của DPPT với user được chia ra làm 3 phần chính :

- *Network System Area :* nằm phía dưới (bottom) của màn hình giao diện , có nhiệm vụ thể hiện cấu hình và trạng thái của hệ thống mạng . Vùng này còn gọi là vùng *System* .
- *User 's Application Area :* thể hiện các module của ứng dụng , mỗi module là một node (từ đây chúng ta sẽ coi module như là một node và hai tên này sẽ được sử dụng với ý nghĩa như nhau trong báo cáo của đề tài này) . Mối quan hệ phân cấp giữa các node được thể hiện trên đồ thị của vùng này tuy nhiên mối quan hệ dữ liệu giữa các node không được thể hiện trên đồ thị . Vùng này còn được gọi ngắn gọn là vùng *Application* .
- *Quick Tool :* gồm nhiều button như *map , source code , create , properties , ...* dùng để thao tác với các node trong vùng Application cũng như với các host trong vùng System.

Ngoài ba phần này DPPT còn có một menu ngang với nhiều chức năng hỗ trợ cho user khi làm việc với công cụ này như save lại vùng application , update lại mạng , hiển thi thông tin về mạng ,về ứng dụng , chức năng hướng dẫn , ... chúng ta sẽ giới thiệu kỹ ở các mục sau.

(xem hình vẽ 2.1.1)



Hình 2.1.1

Sau đây chúng ta sẽ làm quen với một vài tính chất về màu sắc cũng như chức năng của các node cũng như các host trong vùng Application và vùng System .

Một bài toán lớn khi chia ra thành nhiều bài toán nhỏ thì sẽ có nhiều node trên đồ thị của vùng Application . Vì các node được tổ chức theo dạng cây và chỉ có các node là (leaf node) mới là các node thực sự được thực thi trên các host của mạng nên để phân biệt giữa các node lá và các node trung gian cũng như phân biệt trạng thái của các node (đã được map ? , có ở trạng thái zoom hay không ? , node đã được generate chưa ? , ...) thì DPPTthể hiện mỗi loại node bằng một màu sắc khác nhau .

Hiện giờ DPPT dùng 4 màu để thể hiện trạng thái của các node (cũng tương tự cho các host sẽ nói ở sau) : đỏ , xanh (blue) , xám (grey) và tím . Khi chúng ta click mouse vào một node nào đó thì node đó sẽ có màu đỏ (trong đồ thị chỉ có một node màu đỏ và tương tự cho các host) thể hiện trạng thái node đó đang được lựa chọn . Điều này rất quan trong vì hầu hết các button

trong vùng Quick Tool đều hoạt động dựa vào trạng thái có node nào được lựa chọn hay không ? (sẽ nói kỹ trong phần chức năng của các buuton) . Các node lá đều có màu xanh ngoại trừ trường hợp node lá đó ở trạng thái zoom . Màu xám thể hiện cho các node không tích cực tức là các node không đại diện cho một bài toán con nào cả . Các node trung gian trong đồ thị và các node ở trạng thái zoom đều có màu xám . Cuối cùng là màu tím , node nào có màu tím chứng tỏ đã được map lên một host trong vùng System .

Về màu sắc thì có sự tương ứng giữa vùng Application và vùng System : host có màu xám là các host mà DPPT không có quyền sử dụng (xem phần nói về detect host) , host màu đỏ là host đang được lựa chọn , host màu xanh là host mà DPPT có thể sử dụng được còn host màu tím là host mà trên đó có ít nhất một node (module) đã được map lên đó (tức là module đó , bài toán nhỏ đó sẽ được thực thi trên host này) .

Ngoài màu sắc ra thì vùng Application còn có hai ký hiệu nữa cho các node : node nào ở trạng thái zoom ngoài việc có màu xám sẽ còn thêm một chữ 'z' ở giữa node ; node nào đã được generate thì có một chữ 'g' (chú ý rằng có thể có node có hai chữ 'zg') .

2. Chức năng của các Menu , các Button và cụ thể về các vùng trong giao diện với user của DPPT

Trong phần này chúng ta sẽ giới thiệu chi tiết về cách sử dụng phần mềm DPPT gồm chức năng cụ thể về các vùng trong GUI , chi tiết chức năng của các button , các Menu và tóm tắt lại qui trình thao tác trên GUI của một user để giải quyết một vấn đề lớn đã được chia thành nhiều công việc nhỏ hơn .

2.1. Vùng Network System Area

Khi DPPT khởi động thì nó sẽ tự động tìm kiếm các host còn sống trên mạng và hiển thị lên vùng này theo như màu đã nói ở trên : khi detect thấy một host thì DPPT sẽ kiểm tra xem host này có thỏa mãn yêu cầu của DPPT hay không (sẽ nói kỹ trong phần 3) , nếu thỏa mãn thì host sẽ được vẽ trong vùng này với màu xanh (blue) và được gán một số tùy theo thứ tự mà host đó được tìm thấy trong mạng . Ngược lại host sẽ được vẽ với màu xám (grey) . Trong quá trình thao tác với GUI của DPPT thì các host sẽ không được cập nhật lại một cách tự động mà user phải tự làm công việc này qua một chức năng của Menu ngang . Lý do cho điều này là vì mỗi khi cập nhật lại các host thì DPPT sẽ tự động gỡ các module đã được map lên các host và sau đó user phải map lại các module lên các host mới được tìm thấy . Đây là một quá trình khá tốn thời gian nên DPPT không làm tự động (mặc dù DPPT cũng có chức năng map tự động) mà để cho user làm và khi cập nhật lại các host nếu có host đã "chết" (so với lần detect trước) thì host đó cũng được vẽ trong vùng System này nhưng ở một hàng bên dưới hàng thể hiện các host đang còn sống của mạng . Điều này cũng có nghĩa rằng khi đang thao tác với DPPT thì user không hề nhận được cảnh báo gì về việc các host trên mạng có "chết" hay không mà chỉ đến khi cập nhật (update) lại các host trên mạng thì user mới có thông tin về các host đã chết hay còn sống trên mạng .

Khi vẽ một host trong vùng này thì DPPT cũng hiển thị thêm một vài thông tin ngay bên dưới mỗi host (kể cả host đã "chết") : một hàng hiển thị hostname , một hàng hiển thị username đang sử dụng DPPT , một hàng hiển thị địa chỉ IP của host và hàng cuối cùng hiển thị số module (hay số node) đã được map lên host đó . Ngoài ra trên mỗi host còn hiển thị một con số cho biết

thứ tự của host khi nó được tìm thấy lúc DPPT khởi động hoặc khi cập nhật lại các host trên mạng .

Ngoài các đặc điểm này , khi thao tác với mỗi host , DPPT còn cung cấp thêm vài chức năng để biết thêm thông tin hay delete bớt host khỏi cấu trúc dữ liệu , lấy thông tin về mạng, ... và các chức năng này sẽ được giới thiệu trong các mục sau . Và cuối cùng xin nhắc lại rằng nếu chúng ta click vào một host nào đó thì host đó sẽ có màu đỏ thể hiện cho trạng thái đang được lựa chọn của nó .

2.2. Vùng User's Application Area

Vùng này sẽ hiển thị thông tin về ứng dụng (bài toán) đang được xây dựng dựa trên phần mềm DPPT . Từ một bài toán lớn , user có nhiệm phân tích và thiết kế để phân chia bài toán đó ra thành những module nhỏ hơn . Mỗi ứng dụng có thể có nhiều module tùy theo sự thiết kế của các user .

Mỗi module được biểu diễn bằng một node trên đồ thị dạng cây được thể hiện trong vùng này . Vì đồ thị dạng cây nên tạo cho user có một cái nhìn bao quát hơn về ứng dụng mà mình đang thiết kế . Một node trên đồ thị có thể có nhiều node con và giữa các node con và cha của nó có một đường nối với màu sắc giống như màu của node con . Bên cạnh cấu trúc phân cấp của các node và màu sắc của chúng thì để dễ phân biệt các node DPPT còn đánh số các node theo thứ tự root-left-right . Khi DPPT mới khởi động lên hoặc khi muốn tạo ra một ứng dụng mới thì có một node số 0 ,trạng thái tích cực , màu đỏ (đang được lựa chọn) được sinh ra ở góc trên bên trái của vùng Application . Đây là node gốc , nó hoàn toàn có những tính chất giống như các node khác ngoại trừ một điều rằng user *không thể delete node gốc* này (vì một ứng dụng thì ít nhất phải có một module) .

Có hai loại node khác nhau trên đồ thị :

- *Node tích cực :* là những node có màu xanh hoặc màu tím (đã map) trên đồ thị và thường là các node lá ,tuy nhiên khi một node trung gian ở trạng thái active thì các node con của nó sẽ ở trạng thái zoom tức là không còn tích cực nữa . Loại node này còn có thể có chữ 'g' (generate) nhưng không bao giờ có chữ 'z' (zoom) . Để biết cách map một node lên một host hãy xem mục 2.3 .
- *Node không tích cực :* là những node có màu xám trên đồ thị . Những node này chỉ có ý nghĩa trong quá trình thiết kế top-down của user chứ hoàn toàn không còn ý nghĩa trong khi thực thi ứng dụng trên các host của mạng (ít nhất là trong version này của DPPT) .

Các node xét về phương diện của user là một trong những phần quan trọng nhất khi giải quyết bài toán của họ . Có rất nhiều thao tác với các node để tạo ra được một module khả thi trên mạng như soạn thảo code cho node , nhập các properties của node , map node , ... và các thao tác này sẽ được trình bày trong hai mục sau đây .

2.3. Chức năng của các Button - Vùng Quick Tool

Mục này và mục 2.4 là hai mục quan trọng nhất đối với một user của DPPT. Vùng Quick Tool bao gồm những thao tác trên các node của vùng Application để tạo ra một ứng dụng có thể

chạy được trên mạng còn Menu ngang nhằm hỗ trợ thêm cho user về các thông tin có tính khái quát cũng như nhiều tiện ích khác cho phần hệ thống (vùng System) . Cả hai phần này cần được sử dụng để bổ sung cho nhau nếu muốn tạo ra một úng dụng hoàn chỉnh trên công cụ DPPT .

Vùng Quick Tool gồm có 14 button và chúng ta sẽ giới thiệu chức năng của từng button một cách chi tiết cùng với những thông tin có liên quan để có được chức năng đó .

2.3.1. Create Button

Đây là button mà bất cứ một user nào cũng phải sử dụng (tất nhiên nếu ứng dụng của họ được thiết kế có nhiều hơn một module) . Button này có nhiệm vụ sinh ra một node con của node có màu đỏ (node đang được lựa chọn) . Ở đây một lần nữa xin nhắc lại rằng các button của vùng Quick Tool đều làm việc với các node có màu đỏ . Để tao ra một node có màu đỏ thì chúng ta chỉ đơn giản click mouse vào node đó và tại một thời điểm chỉ có một node màu đỏ trong toàn bộ cây ứng dụng trong vùng Application .

Ví dụ user muốn thiết kế một node con cho một node nào đó trong đồ thị thì chỉ cần thực hiện theo hai bước :

- Click chuột vào node muốn sinh ra node con , node này sẽ đổi qua mầu đỏ nếu trước đó nó có mầu khác .
- Bấm vào Create button thì sẽ có thêm một node con xuất hiện ở hàng dưới của node này đồng thời có một đoạn màu xanh nối từ node con đó tới node này là node cha của nó . Nếu trước khi sinh ra node con mà node này có màu xanh (node tích cực) thì sau khi sinh con node này sẽ có màu xám .

Tuy nhiên có một vài điều kiện để cho phép một node có thể sinh thêm một con :

- Node không được có màu tím : tức là node chưa được map lên một host nào của hệ thống cả .
- Node không được có chữ ‘z’ : tức là node không được ở trạng thái zoom .

Sau đây là thủ tục được gọi mỗi khi user bấm vào Create button để các độc giả tham khảo nhanh nếu quan tâm (tất cả Source Code của chương trình được trình bày cụ thể trong phần 6 của bản báo cáo) :

```
void Create_Node_on_Tree(NODE_POINTER current)
{
        if ((current!=NULL)&&(b==TRUE))
        {
                if ((current->get_selected()==TRUE)&&(current->get_map()==FALSE)
                   &&(current->get_zoom()!=TRUE))
                {
                        NODE_POINTER new_node=new NODE();
                        current->set_active(FALSE);
                        new_node->parent=current;
```

```
                    new_node->set_y(current->get_y()+1);
                    if (current->child==NULL)
                        current->child=new_node;
                    else
                    {
                        NODE_POINTER child=current->child;
                        while (child->peer!=NULL)
                            child=child->peer;
                        //them vao list con cua current
                        child->peer=new_node;
                    }
                    b=FALSE;
                }
                else
                {
                    Create_Node_on_Tree(current->child);
                    Create_Node_on_Tree(current->peer);
                }
        }
}
void CreateNode()
{
        b=TRUE;
        Create_Node_on_Tree(root_node);
        DrawAllNode();
}
```

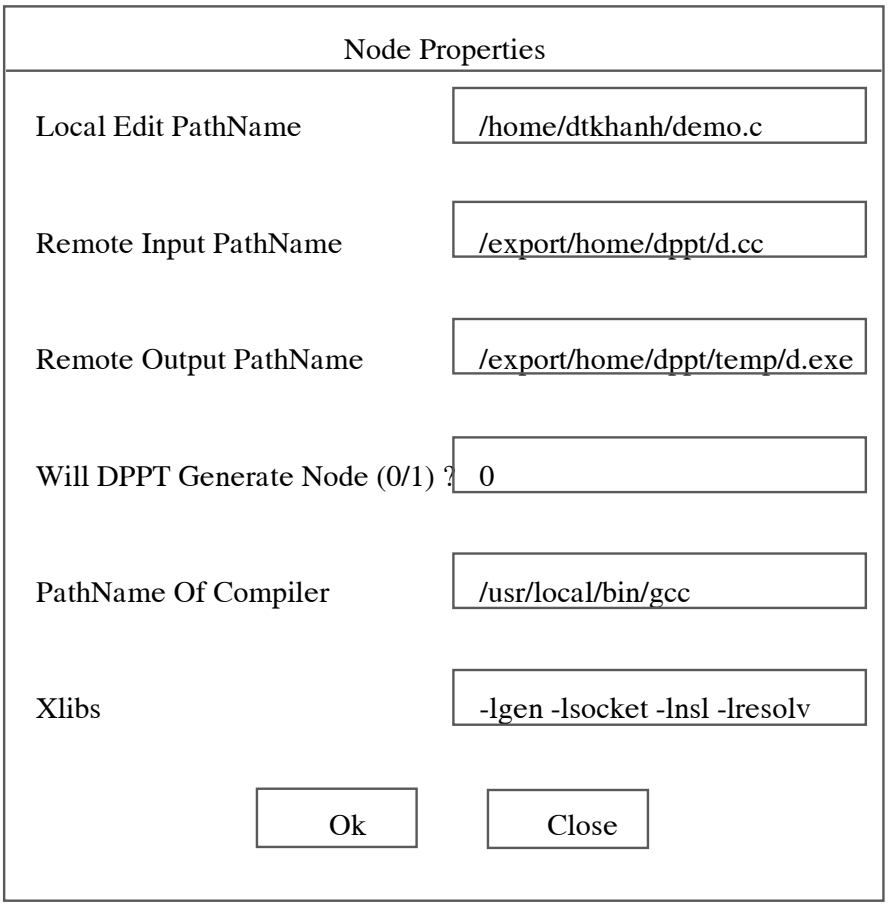
2.3.2. Properties Button

Bất cứ một node tích cực nào (màu xanh hoặc tím) muốn trở thành một chương trình có thể chạy được trên các host thì phải nhập vào các properties để đặc tả cho node đó .

Để nhập vào properties của một node chúng ta thực hiện các bước sau :

- Click chuột vào node cần nhập properties : node sẽ đổi sang màu đỏ .
- Nếu node đó là node tích cực thì DPPT sẽ hiển thị một bảng các tính chất của node (property table of node) và user nhập vào theo hướng dẫn dưới đây . Ngược lại thì sẽ có những cảnh báo khác để báo cho user biết .

Sau đây là hình dạng bảng properties của một node tích cực :



Hình 2.3.2.1

- *Local Edit PathName :* ở đây user nhập vào tên file (có thể cần cả đường dẫn đầy đủ) để chỉ ra file nào (và ở đâu) chứa chương trình nguồn cho node này ở trên local host (tức là host mà DPPT đang chạy trên đó) . Đây là một mục quan trọng nhất của properties vì các thao tác của DPPT như remote compile , soạn thảo module , ... đều có tham khảo đến mục này của Node Properties . Mặc dù mỗi mục trong Node Properties đều có gia trị default khi node được tạo ra nhưng các giá trị này có thể không đúng với ý muốn của user do đó user vẫn cần phải nhập lại để đảm bảo chương trình chạy đúng ý muốn của mình .

Ở ví dụ trên thì file cần soạn thảo cho node có tên là *demo.c* và ở trong thư mục */home/dtkhanh* .

- *Remote Input PathName :* khi muốn chạy một module trên một remote host trong mạng thì DPPT sẽ đưa nguyên cả source code (kèm theo các include file) sang remote host và dùng compiler có ở remote host để dịch và tạo ra một chương trình thực thi được trên remote host . Điều này làm cho DPPT có thể chạy được trên nhiều loại máy khác nhau được nối thành mạng mà không nhất thiết các máy trên mạng phải là các máy đồng nhất (homogeneous computer) . Chính vì vậy nên mục này user sẽ phải nhập vào tên của file trên remote host mà file đó là một bản copy

của Local Edit PathName đưa qua để compiler trên máy từ xa . Nếu module này được user dự định cho chạy trên local host thì user không cần nhập mục này . Ở ví dụ trên thì file demo.c sẽ được chép qua remote host với tên d.cc trong thư mục /export/home/dppt .

- *Remote Output PathName :* khi dịch ra file thực thi thì trình biên dịch cần biết file đích (dù là remote hay local host) và mục này là nơi mà user nhập vào tên file đích (có thể cả đường dẫn) cho trình biên dịch . Ở ví dụ trên file thực thi là d.exe .
- *Will DPPT Generate Node (0/1) ? :* mục này user chỉ có thể nhập là 0 hoặc 1 . Nếu nhập không đúng thì DPPT sẽ thông báo ngay tại mục này . Nếu user nhập vào số 0 thì code cho module này user phải viết thành một chương trình hoàn chỉnh và chỉ cần biên dịch là có thể chạy được . Nếu user nhập vào số 1 thì code cho node này user không cần viết hoàn chỉnh , ở đây tức là *không cần có hàm main()* nhưng phải có một hàm dạng *void* DPPT_RUN()* và sau đó DPPT sẽ tự sinh code hoàn chỉnh cho node này khi user lựa node này và bấm Generate button . Sau đó node này có thể được biên dịc và chạy bình thường như các node khác .
- *PathName Of Compiler :* mục này chỉ ra trình biên dịch mà user sẽ dùng cho node này để tạo ra file thực thi . Với tính chất này chúng ta thấy DPPT có thể dùng được nhiều trình biên dịch . Thực tế tác giả đã sử dụng các trình biên dịch trong một ứng dụng demo như CC , g++ , c++ , gcc .
- *Xlibs :* giá trị default được ghi như trong bảng trên . Ở đây user nhập vào các thư viện cần cho module của mình như -lthread , -lnsl , -lsocket , ...

Ngoài việc sử dụng được nhiều compiler khác nhau , DPPT còn sử dụng được cả makefile . Để làm được điều này thì đơn giản user chỉ cần nhập vào đường dẫn đến thư mục có chứa makefile trong phần Remote Input PathName (ví dụ /home/dtkhanh/) và nhập chuỗi "make" vào phần PathName Of Compiler . Trong chương trình demo xử lý ảnh Mandelbrot (có đĩa kèm theo bản báo cáo) tác giả Đặng Trần Khánh đã sử dụng makefile , độc giả có thể xem và thử trên mạng để tham khảo .

Nói chung DPPT đã đưa ra một giải pháp khá hợp lý cho phép các user có thể *tùy ý* sử dụng các compiler mà họ sẵn có . Đây cũng là một điểm mạnh của DPPT .

Sau khi nhập xong các properties cho một node , user bấm Ok để cập nhật lại thông tin về properties cho node đó rồi bấm Close để đóng property table và trở về hoặc *cứ để bảng property đó và chọn một node khác để nhập properties cho các node cho đến khi bấm Close*. Chương trình nguồn của phần này độc giả có thể tham khảo ở phần 6 vì nó được xử lý khá tinh tế và khá nhiều và vì thế tương đối dài cho nên tôi không tiện đưa ra ở đây .

2.3.3. Source Code Button

Đây cũng là một button mà user phải thường xuyên sử dụng khi làm việc với DPPT . Sau khi đã thiết kế và phân chia bài toán ra thành nhiều module thì mỗi module đại diện cho một bài toán nhỏ hơn và mỗi module theo quan điểm của DPPT là một chương trình hoàn chỉnh có thể thực thi (về mặt cú pháp của ngôn ngữ lập trình) . Do đó muốn viết code cho các module này thì user phải sử dụng Source Code button . Để soạn thảo cho một module thì user lựa chọn node ứng với module đó (sẽ có màu đỏ) rồi bấm vào button Source Code . Lúc này DPPT sẽ gọi trình soạn thảo "textedit" với một tham số là Local Edit PathName trong Node Properties. Nếu file đã tồn

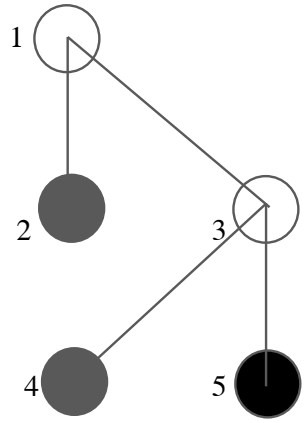
tại thì DPPT mở file cho user sửa đổi còn ngược lại thì một file mới sẽ được tạo ra với tên nhập vào từ Local Edit PathName trong Properties button .

Chú ý rằng nếu không có node nào được lựa chọn mà user vẫn bấm vào Source Code thì DPPT vẫn chạy "textedit" nhưng với một cửa sổ textedit trống , ở đây user vẫn có thể soạn thảo bình thường cho các source code file hoặc các include file định nghĩa các hằng , cấu trúc dữ liệu , ... của module .

2.3.4. Zoom Out Button

Zoom Out và Zoom In button là hai button đặc biệt tạo ra sự thoải mái , tiện lợi cho các user khi phân chia bài toán thành các module nhỏ hơn .

Để hiểu chức năng của button này chúng ta hãy xem hình vẽ 2.3.4.1 . Giả sử node 1 đã được chia ra thành hai node 2 và 3 , node 3 lại được chia thành các node 4 và 5 . Như vậy các node 1 và 3 là các node không tích cực . Lại giả sử node 5 đã được map lên một host nào đó trong vùng System . Tuy nhiên sau khi phân chia như vậy thì user lại thấy rằng sự phân chia của mình là không cần thiết và node 1 không cần phải chia ra thành các bài toán con nhỏ hơn nữa và user muốn node 1 trở thành node tích cực nhưng vẫn muốn lưu lại hình dạng của đồ thị vì sau đó có thể họ sẽ lấy lại ý tưởng phân chia như cũ (!?) . Để làm được điều đó , đơn giản user chỉ cần click vào node 2 hoặc node 3 (sẽ có màu đỏ) rồi bấm vào Zoom Out button thì node 1 sẽ trở thành node tích cực , các node 2,3,4 và 5 sẽ trở thành các node không tích cực có màu xám và chữ 'z' ở giữa node (riêng node 1 có màu xanh và chữ 'z') . Chú ý rằng node 5 đã được map thì DPPT sẽ tự động gỡ nó ra khỏi host mà nó được map lên đó .



Hình 2.3.4.1

Ngoài ra điều kiện để thực hiện được chức năng này là node cha của node được lựa chọn phải không được ở trạng thái zoom (anode->parent->get_zoom()!=TRUE) và node được lựa chọn phải khác node gốc (root node) , nếu không thì DPPT sẽ không làm gì cả !!!. Ở đây chú ý rằng một node ở trạng thái zoom vẫn có thể tích cực (như node 1 sau thao tác ở trên) .

Sau đây là đoạn code cho thủ tục sẽ được gọi (thủ tục Zoom_Out()) mỗi khi user bấm vào Zoom Out button , dành cho các độc giả quan tâm có thể tham khảo nhanh để hiểu rõ hơn về

phần này (và một lần nữa xin nhắc lại rằng toàn bộ source code của luận án được trình bày trong phần 6) :

```
void zoom_out_subnode(NODE_POINTER anode)
{
        if (anode!=NULL)
        {
                if (anode->get_map()==TRUE)
                {
                        anode->set_selected(TRUE);
                        b=TRUE;
                        detach_selected_node(anode);
                        anode->set_active(FALSE);
                }
                else
                        anode->set_active(FALSE);
                anode->set_zoom(TRUE);

                zoom_out_subnode(anode->child);
                zoom_out_subnode(anode->peer);
        }
}

void zoom_out_on_tree(NODE_POINTER anode)
{
        if (anode!=NULL)
        {
                if ((anode->get_selected()==TRUE)&&
                   (anode->parent->get_zoom()!=TRUE))
                {
                        anode->set_selected(FALSE);     //de thay su doi mau

                        anode=anode->parent;                    //xet node cha
                        anode->set_active(TRUE);
                        anode->set_zoom(TRUE);

                        zoom_out_subnode(anode->child);
                }
                else
                {
                        zoom_out_on_tree(anode->child);
                        zoom_out_on_tree(anode->peer);
                }
        }
}
```

```
void Zoom_Out()                    //thu nho lai
{
        zoom_out_on_tree(root_node->child);   //khong cho zoom_out root_node
        DrawAllNode();
        DrawHostSystem();
}
```

2.3.5. Zoom In Button

Chức năng của button này hoàn toàn ngược lại với Zoom Out button . User chỉ cần click vào node 1 trong đồ thị ví dụ ở trên rồi bấm vào Zoom In button thì node 1,3 lại là các node không tích cực còn node 2,4,5 lại là các node tích cực . Chú ý rằng user phải click vào node 1 còn nếu click vào node 2,3,4 hay 5 thì DPPT không làm gì cả lý do là vì node cha của chúng là node 1 đang còn ở trạng thái zoom thì các con không thể ở trạng thái không zoom được .

Ở đây cũng chú ý rằng nếu node 1 đã được map lên một host nào đó thì khi Zoom In , DPPT cũng sẽ tự động gỡ (detach) node đó ra khỏi host mà nó được map lên .

2.3.6. Generate Button

Đây có lẽ là button ít được sử dụng nhất của DPPT trong phần Quick Tool . Như đã nói trong phần Properties button , nếu một node có cờ generate = 1 thì source code cho node đó không cần là một chương trình C (hay C++) hoàn chỉnh , *ở đây là không cần có hàm main()* , nhưng yêu cầu trong khi soạn thảo source code của node này thì phải có một hàm có dạng *void* DPPT_RUN()* để khi bấm vào button này thì DPPT sẽ sinh code hoàn chỉnh cho node đang được lựa chọn (thành một chương trình có hàm main()) .

Ví dụ khi cờ generate = 1 trong phần properties của node(j) và source code của node(j) có dạng như sau :

```
...

void* DPPT_RUN()
{
        ...
}

...
```

thì khi click vào node(j) rồi bầm vào Generate button thì đầu tiên user sẽ thấy trên vùng Application node(j) có thêm một chữ ‘g’ ở giữa node (và nếu user không biết cách reset lại trạng thái của node(j) thì nó sẽ luôn có chữ ‘g’ đó) , thứ hai khi lựa node(j) và bấm vào Source Code button thì user sẽ thấy code của node(j) bây giờ sẽ là :

```
//*****************************************
```

```
//    User phai co ham void* DPPT_RUN()
//*****************************************

void* DPPT_RUN();

int main(int argc,char** argv)
{
        DPPT_RUN();

        return 0;
}

...

void* DPPT_RUN()
{
        ...
}

...
```

và lúc này chúng ta sẽ có một chương trình C (C++) hợp lệ để biên dịch và thực thi trên host mà DPPT có quyền (là những host màu xanh hoặc màu tím) .

Cuối cùng như chúng ta đã biết sau khi generate cho một node thì user vẫn có khả năng sửa đổi source code của node và không có lý do gì ràng buộc nếu họ xóa đi phần code mà DPPT đã generate và yêu cầu DPPT hãy generate lại cho họ đoạn code mà họ vừa xóa ngoại trừ một điều là user phải làm sao cho chữ 'g' trên node biến mất trước khi họ có thể yêu cầu DPPT làm điều đó . Để làm được điều này user chỉ cần click vào node đó một lần nữa và bấm vào Generate button thì chữ 'g' sẽ biến mất (lúc này trong cấu trúc dữ liệu thì cờ done được reset = 0) . Sau đó làm lại bước generate node một lần nữa thì user sẽ thu được kết quả mong muốn .

Đoạn code cho chức năng này của DPPT cũng khá thú vị nên chúng tôi cũng đưa ra đây để các độc giả quan tâm có thể tham khảo nhanh :

```
void Generate_Node(NODE_POINTER anode)
{
        if ((anode->get_generate()==TRUE)&&(anode->get_done()==FALSE))
        {
                char str[MAX_PATHNAME];
                int count=0;

                sleep(1);       //delay
```

```
                sprintf(str,"cat generate.dppt %s > %s.%d",anode->get_edit_file(),TEMP_FILE,anode->get_id());

                while (system(str)<0)
                {
                        cout<<"Error when generating ... Waiting..."<<endl;
                        sleep(3);
                        count++;
                        if (count>9)
                        {
                                cout<<"Bye.."<<endl;
                                return;
                        }
                }

                sprintf(str,"mv %s.%d %s",TEMP_FILE,anode->get_id(),
                        anode->get_edit_file());
                while (system(str)<0)
                {
                        cout<<"Error when generating ... Waiting..."<<endl;
                        sleep(3);
                        count++;
                        if (count>9)
                        {
                                cout<<"Bye.."<<endl;
                                return;
                        }
                }
                anode->set_done(TRUE);          //compiled NODE
                cout<<endl<<"Node "<<anode->get_id()<<" : Generated"<<endl;
        }
        else
        {
                if (anode->get_done()==TRUE)
                {
                        anode->set_done(FALSE);
                        cout<<"Node "<<anode->get_id()<<" : Reset DONE=FALSE"<<endl;
                }
                else
                {
                        cout<<endl<<"Node "<<anode->get_id()<<" : Generation=FALSE"<<endl;
                        anode->set_done(FALSE);   //xoa done flag neu co
                }
        }
}
```

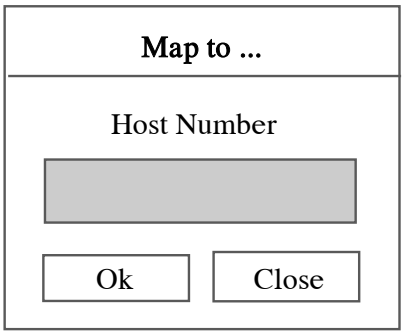
2.3.7. Map Button

Trong phần này ngoài hướng dẫn cách sử dụng Map button chúng ta cũng sẽ được giới thiệu cách sử dụng mouse để làm được công việc này một cách nhanh chóng hơn .

Như chúng ta đã biết , bài toán lớn của user được chia ra thành nhiều bài toán nhỏ hơn biểu diễn bằng các node tích cực trên đồ thị trong vùng Application . Để tận dụng sức mạnh của mạng chúng ta sẽ cho các bài toán nhỏ này được thực thi (chạy) trên các host của mạng . Các host này được DPPT tìm thấy và hiển thị trong vùng System và chỉ có các host màu xanh khi chưa có node nào được map , nếu có node đã được map đúng thì thêm các host màu tím (node map đúng là các node được map lên các host mà lần đầu tiên nó hiện ra trong vùng System thì nó có màu xanh) . Các host có màu xám DPPT chỉ detect để tham khảo chứ user của DPPT hoàn toàn không thể sử dụng bất cứ một tài nguyên nào của các host này trong quá trình giải quyết bài toán của mình (phần nói về detect host trong phần 3 sẽ đề ccập chi tiết đến vấn đề này) .

Để map được các node lên các host trong vùng System bằng Map button chúng ta làm như sau :

- Click vào node cần map (phải là node tích cực và chưa được map) .
- Bấm chuột vào Map button chúng ta sẽ thấy một cửa sổ hiện ra như hình vẽ 2.3.7.1 đưới đây rồi nhập số thứ tự của host vào vùng hostnumber và bấm Ok thì node sẽ được map lên host có số thứ tự đó .



Hình 2.3.7.1

Ở đây chúng ta có một cách làm nhanh hơn để map các node lên một host bằng mouse một cách rất đơn giản như sau : chúng ta chỉ cần bấm chuột vào node cần map nhưng không nhả ra mà cứ giữ như vậy rồi kéo đến host cần map trong vùng System và nhả chuột ra là xong (mouse drap and drop function) . Nói chung thì đa số các thao tác quan trọng và cần phải làm nhiều lần khi làm việc với DPPT đều được hỗ trợ bằng mouse tạo cho các user sự thoải mái và thân thiện khi xây đựng bài toán của mình trên Tool này . Trong các mục sau chúng ta sẽ được giới thiệu thêm về các thao tác khác với mouse rất tiện lợi khi là một user của phần mềm DPPT .

2.3.8. Detach Button

Một khi user đã map (hoặc được DPPT map tự động như sẽ nói sau này) một node lên một host nhưng vì lý do nào đó họ lại không muốn cho node đó chạy trên host này mà muốn nó được thực thi trên một host khác thì họ phải map node lên host khác nhưng để làm được việc đó thì công việc đầu tiên mà họ phải làm là gỡ node ra khỏi host mà nó được map lên .

Để gỡ (detach) *một node* ra khỏi host thì user làm hai động tác sau :

- Click mouse vào node cần được detach (tất nhiên node đó phải có màu tím) .
- Bấm vào Detach button .

Ở phần detach node này , DPPT còn cung cấp hai chức năng tiện lợi thao tác bằng mouse như sau :

- Gỡ *một node* hay một *sub-graph* bằng mouse : Drap and Drop node cần gỡ vào waste button (được vẽ với biểu tượng một thùng rác) ta sẽ được kết quả tương tự như sử dụng detach button ở trên . *Chú ý rằng nếu node chưa được map mà dùng chức năng này thì node sẽ bị xóa khỏi graph trong vùng Application cũng như trong cấu trúc dữ liệu của DPPT .* Nếu muốn detach các con cháu của một node thì chúng ta cũng dùng mouse tương tự như trên nhưng node cần drap and drop là một node trung gian . Gỡ một sub-graph và gỡ một node bằng mouse thực ra thì thao tác như nhau nhưng công dụng khác nhau rất nhiều và có thể dẫn tới tốn thời gian cho user phải map lại các node nếu như detach sai một sub-graph nên phải sử dụng cẩn thận .
- Gỡ các node được map lên một host : để detach tất cả các node đã được map lên một host (map đúng hay sai host thỏa yêu cầu của DPPT đều được), user chỉ cần drap & drop host đó vào "thùng rác" của DPPT (chứ không phải thùng rác của Unix) . *Ở đây cũng cần chú ý rằng nếu host không có node nào được map lên (không có màu tím) thì nó sẽ bị xóa khỏi vùng hiển thị System cũng như trong cấu trúc dữ liệu của DPPT (!?) .*

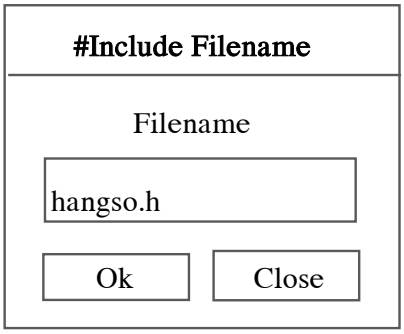
Source cho các chức năng này độc giả có thể tham khảo trong phần 6 của bản báo caco này , các thủ tục có liên quan là : Detach_Node , detach_selected_node, detach_subgraph , Delete_Host,delete_host_by_mouse .

2.3.9. #Include Button

Với button này DPPT cho phép các user chỉ ra các include file do user viết (chứ không phải của C hay C++ Compiler) và được sử dụng cho node nào đó trong vùng Application . Ví dụ node 1 có dùng hằng MIN được định nghĩa trong file *hangso.h* chẳng hạn thì user phải chỉ ra cho DPPT (và DPPT sẽ chỉ ra cho C Compiler) biết node 1 có những include file thêm vào như thế nào bằng cách sau :

- Click vào node 1 .

- Bấm vào #Include button thì có một cửa sổ nhập hiện ra và chỉ có một dòng nhập . User nhập vào đó chuỗi *hangso.h* và click vào Ok để cập nhật data cho node 1 sau đó là Close để đóng cửa sổ nhập . Sau đây là hình dạng của cửa sổ nhập :



Hình 2.3.9.1

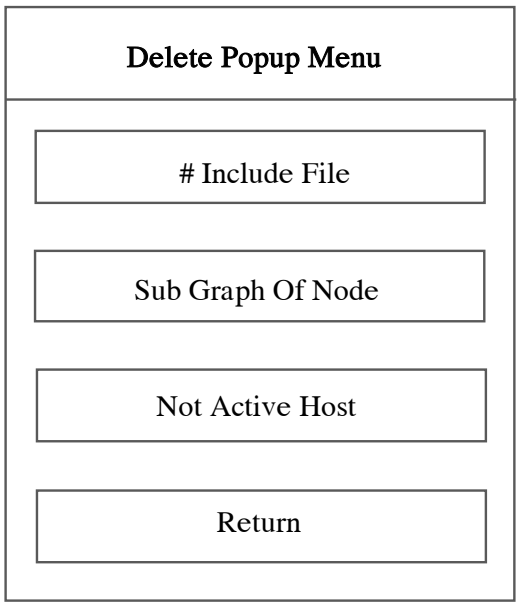
Ở đây cũng cần chú ý thêm một điều rằng nhiều khi DPPT không thể tìm thấy include file *hangso.h* là do biến môi trường trong file *.profile* chưa được thiết lập nên user phải **nhập pathname đầy đủ cho file hangso.h** (ví dụ như /home/student/dtkhanh/demo1/hangso.h) .

Ngoài ra các user cũng cần lưu ý thêm một điều nữa là nếu họ tự động thêm dòng code #include "hangso.h" vào trong source code của node 1 thì khi node 1 được cho chạy ở remote host nó sẽ *không thể được biên dịch đúng* vì tên file cho một node ở remote host đã được DPPT thay đổi kể cả các include file để đảm bảo không bị đụng độ giữa các user đang dùng DPPT (sẽ nói kỹ ở phần compile) . Xin nhắc lại rằng DPPT là một phần mềm multiuser nên các user cần phải theo đúng các hướng dẫn của DPPT để đảm bảo chương trình được chạy đúng .

2.3.10. Delete Button

Button này được dùng cho cả các node trong vùng Application và cho các host trong vùng System . Ngay tên của button cũng đã nói lên phần nào chức năng của nó .

Button này có một công dụng và sức mạnh bằng nhiều tác vụ khác cộng lại (bao gồm cả các thao tác bằng chuột) . Khi user bấm vào button này thì một popup menu hiện ra có dạng như hình vẽ 2.3.10.1 sau đây :



Hình 2.3.10.1

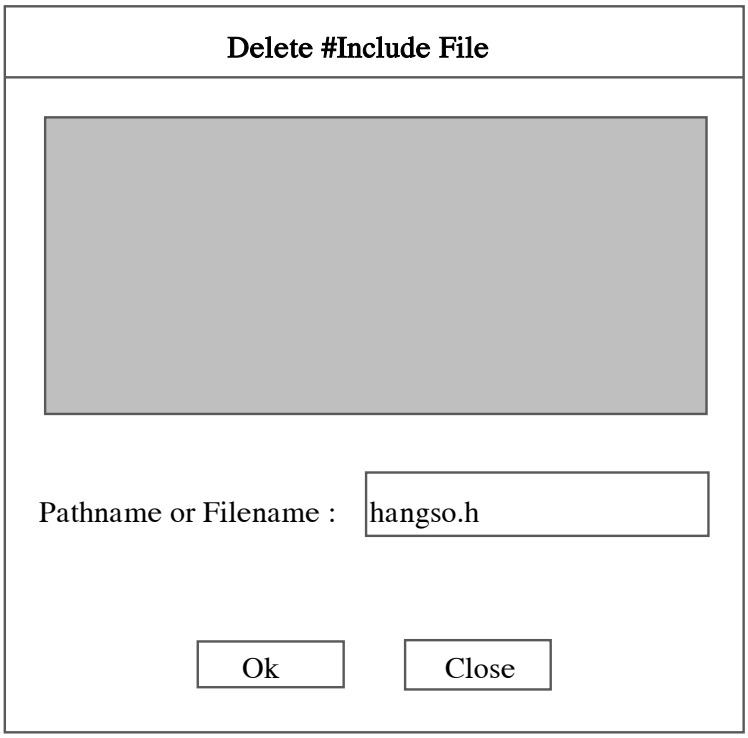
Sau đây là các chức năng của các mục trong Popup Menu này :

- *# Include File :* Trong ví dụ ở trên , nếu node 1 không cần dùng tới hằng số MIN nữa thì cũng không cần include file *hangso.h* nữa . Để delete file *hangso.h* trong cấu trúc dữ liệu của node 1 thì click vào node 1 và bấm vào mục này chúng ta sẽ thấy một cửa sổ nhập như hình 2.3.10.2 . Phía trên của cửa sổ nhập sẽ hiển thị các include file của node 1 trong đó có file *hangso.h* và user nhập tên file cần xóa là *hangso.h* (có thể cả đường dẫn nếu như trong vùng hiển thị có hiển thị nó) vào vùng Pathname or Filename và bấm Ok để chấp nhận xóa file sau đó bấm Close để đóng cửa sổ nhập .

Nếu nhập tên file sai thì DPPT không cảch báo nhưng vùng hiển thị không có gì thay đổi còn nếu nhập đúng thì vùng hiển thị sẽ hiển thị ra các include file còn lại của node (ở ví dụ này là node 1) .

Nếu có node 2 cũng muốn xóa đi include file *hangso.h* thì không cần bấm Close mà chỉ bấm Ok cho node 1 sau đó click vào node 2 rồi lại bấm Ok lần nữa , ... Quá trình này có thể lặp lại đến khi user bấm vào Close trong cửa sổ nhập .

(xem hình vẽ 2.3.10.2 dưới đây)



Hình 2.3.10.2

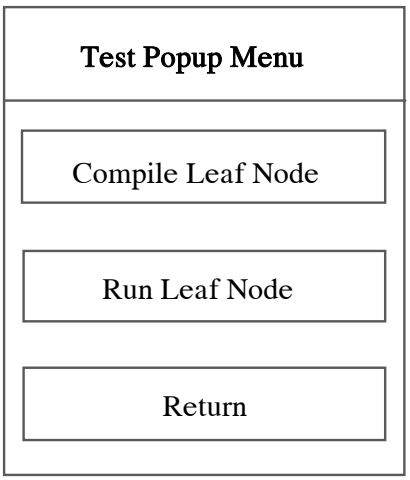
- *Sub Graph Of Node :* Khi user muốn delete một node hay một sub-graph bất kể là node đó đã được map hay chưa và *bất kể trong sub-graph có node nào được map hay chưa* thì chỉ cần click vào node (node lá hoặc node trung gian cho sub-graph) rồi click vào mục này . Khi user bấm vào mục này thì kể cả các node ở trạng thái zoom cũng sẽ bị xóa nhưng có một điều kiện là node cha của nó phải không được ở trạng thái zoom (vì nếu như vậy thì có thể sẽ dẫn tới một node không có con màvẫn ở trạng thái zoom là không phù hợp với quan điểm của DPPT) .
- *Not Active Host :* Vì khi detect các host trong hệ thống thì DPPT hiển thị tất cả các host mà nó tìm thấy kể cả các host mà chương trình của user không thể chạy ở trên đó (DPPT không có quyền chạy chương trình trên host đó) . Các host đó sẽ có màu xám và cũng chiếm một phần bộ nhớ của hệ thống cũng như làm cho phần hiểm thị trong vùng System rất khó nhìn (vì có quá nhiều host) . Nếu user không muốn nhìn thấy các host này nữa và để giải phóng bớt bộ nhớ thì họ click vào mục này , lúc đó vùng System chỉ hiển thị các host tích cực tức là các host mà user có thể map các node lên để thực thi , các host này có màu xanh . Ở đây cần chú ý rằng khi detect các host trong hệ thống thì DPPT cũng nhận dạng các host đã chết so với lần detect trước và hiển thị các host này ở hàng dưới của các host mới tìm thấy . Khi click vào mục này thì DPPT cũng delete luôn các host đã chết trong cấu trúc dữ liệu của mình .
- *Return :* Khi click vào mục này thì Delete Popup Menu sẽ đóng lại . Tất cả các Popup Menu của DPPT đều có mục Return và chức năng đều như nhau là đóng

Popup Menu lại do đó từ đây về sau chúng ta sẽ không giới thiệu lại chức năng của mục return nữa .

2.3.11. Test Button

Đây là phần rất quan trọng của DPPT software , độc giả nên dành thời gian để đọc kỹ mục này và các mục có liên quan ở phần 3 nhằm hiểu được cách DPPT compile và thực thi các node trong ứng dụng của mình để từ đó có thể tránh được nhiều lỗi không đáng có khi làm việc với DPPT . Tất nhiên nếu chỉ là một user và không quan tâm về hệ thống bên dưới thì độc giả chỉ cần đọc mục này để biết các chức năng tổng quát của DPPT và không cần đọc phần 3 của báo cáo này . Sau đây chúng ta sẽ giới thiệu chức năng của Test button .

Button này dùng để kiểm tra (test) mỗi node (nếu có thể) trước khi cho cả ứng dụng đồng thời được thực thi . Đây là bước sơ khởi và là tiền đề cho ý tưởng về một *debuger* của DPPT trong tương lai . Khi user muốn compile hay chạy thử một node nào đó thì họ click vào node đó rồi click vào button này thì một Popup Menu sẽ hiện ra như hình 2.3.11.1 sau :



Hình 2.3.11.1

Sau đây là chức năng của các mục trong Popup Menu trên :

- *Compile Leaf Node :* khi user click vào mục này thì DPPT sẽ tiến hành compile node đã được lựa chọn . Lúc đó DPPT gọi hàm có prototype dạng *void Compile_LeafNode()* và hàm này sẽ gọi hàm *void Compile_Selected_Node (NODE_POINTER anode)* có đoạn code ngắn gọn như sau :

```
void Compile_Selected_Node(NODE_POINTER anode)
{
        if ((anode!=NULL)&&(b==TRUE))
                if ((anode->get_selected()==TRUE)&&
                   (anode->get_active()==TRUE))
                {
```

```
                cout<<endl<<"Starting... Compile Selected Node : ";
                cout<<anode->get_id()<<endl;
                Compile_Node(anode);
                b=FALSE;
        }
        else
        {
                Compile_Selected_Node(anode->child);
                Compile_Selected_Node(anode->peer);
        }
}

void Compile_LeafNode()
{
        b=TRUE;
        Compile_Selected_Node(root_node);
}
```

Như chúng ta thấy , hai hàm này xác định được node nào đang được user lựa chọn và **gọi một hàm rất quan trọng** là hàm *Compile_Node(anode)* để xử lý các công việc còn lại của quá trình compile như xác định host mà node được map , file source code , các include file , trình biên dịch được sử dụng , tên file thực thi , báo lỗi (nếu có) , ... sẽ được chúng ta bàn đến trong một mục riêng ở phần 3 của báo cáo vì đây là một phần rất quan trọng của DPPT , làm nền tảng cho việc compile và thực thi một ứng dụng được xây dựng dựa trên phần mềm DPPT .

- *Run Leaf Node :* thực thi node này sau khi nó đã được biên dịch thành công . *Việc một node có thể chạy (run) một cách độc lập hay không là công việc của user (programmer) phải xác định , DPPT không can thiệp vào công việc này .*

2.3.12. Quick Help Button

DPPT cũng có chức năng help trên menu ngang , tuy nhiên nếu muốn xem nhanh chức năng của các button để thao tác với DPPT thì user chỉ cần click mouse vào button này thì sẽ có một cửa sổ với những hướng dẫn , giới thiệu ngắn gọn về các button được hiện ra cho user tham khảo nhanh . Click vào mục Close trong cửa sổ thì cửa sổ quick help đó được đóng lại .

2.3.13. Done All Button

Button này có biểu tượng một người đang chạy và có chức năng rất dễ hiểu đứng về phương diện của user , khi user bấm vào button này thì nó thực hiện các bước sau :

- DPPT kiểm tra xêm node nào trên đồ thị chưa được map thì sẽ **mặc nhiên cho chạy trên local host .** Chú ý rằng DPPT có chức năng map tự động các node chưa được map **lên các host trong toàn mạng** (chứ không phải chỉ có local host) nhưng ở đây tác giả không dùng chức năng này vì một lý do đã gặp trong thực tế khi tác giả

xây dựng DPPT software : các host trong mạng mà DPPT có quyền thực thi nhưng lại có các acount với tên khác nhau nên mỗi khi map node qua host khác thì phải sửa lại Node Properties (xem phần 3 để biết tại sao acount khác tên trên các host thì phải sửa lại properties khi map) , chính vì thế nên không thể gộp chức năng map tự động vào đây (ít nhất là trong version đầu tiên này của DPPT) .

- DPPT kiểm tra xem node nào trên đồ thị cần generate (xem phần properties button để biết khi nào một node cần generate và các thông tin liên quan) và chưa được generate thì sẽ tự động generate cho node đó .
- DPPT sẽ compile toàn bộ các node lá trong toàn bộ đồ thị với compiler được chỉ ra trong property table của mỗi node .
- DPPT sẽ lần lượt chạy các node lá (tức là chạy toàn bộ ứng dụng) theo thứ tự từ **trái sang phải** của đồ thị .

Sau đây là Source Code cho hàm Done_All() : được goi mỗi khi user bấm vào button có hình người này :

```
void Done_All()
{
        cout<<endl<<"Start All..."<<endl;    //first notice
        sleep(1);

        cout<<"Automatic Mapping..."<<endl;
        Map_All(root_node);          //tu dong map neu chua map
        sleep(1);

        cout<<"Generation Checking..."<<endl;
        Generate_Graph();            //tu dong generate du co hay chua
        sleep(1);

        cout<<"Compiling..."<<endl;
        Compile_Graph();             //compile again if compiled

        cout<<endl<<"Testing ! Wait 5 seconds, please..."<<endl;
        sleep(5);                    //delay

        Run_Application();           //chay Application

        sleep(3);                    //delay

        cout<<endl<<"Ok...Finished All !"<<endl;  //last notice
}
```

Còn đây là Source Code của hàm Run_Application() để thực thi (run) các node lá trên đồ thị từ trái sang phải :

```
void Pre_Run_Application(NODE_POINTER anode)
{
        if (anode!=NULL)
        {
                if (anode->get_active()==TRUE)
                {
                        Run_Node(anode);
                        Pre_Run_Application(anode->peer); //vi node ACTIVE khong
                                                          //co node con ACTIVE
                }
                else
                {
                        Pre_Run_Application(anode->child);
                        Pre_Run_Application(anode->peer);
                }
        }
}

void Run_Application()
{
        Pre_Run_Application(root_node);
}
```

2.3.14. Waste Button

Tới thời điểm hiện thời thì khi bấm vào button này user sẽ không thấy có gì thay đổi (tất nhiên là chỉ nói về quan điểm của user còn trong DPPT thì có đặt lại một vài cờ cho các tác vụ của riêng nó mà chúng ta không cần đề cập ở đây) . *Button này chỉ là nơi để nhả chuột trong các tác vụ drap & drop mouse của DPPT như delete node, delete host , detach node ,...* Tất nhiên nếu user không nhả chuột đúng vị trí này thì các tác vụ đó sẽ không được hoàn thành .

2.3.15. Các chức năng phụ khác

Có một chức năng thao tác với các node trong vùng Application nhưng không được đề cập đến trong các mục trên vì nó không hề liên quan đến các button trong vùng quick tool mà chỉ là thao tác bằng chuột trong vùng Application : khi user muốn chuyển một node hay cả một sub-graph từ node này sang thành con của node khác thì user drop & drap node hay sub-graph đó tới vị trí mới trong đồ thị ứng dụng . Chức năng này cũng nhằm mục đích tạo ra sự thuận tiện cho user khi mô hình hóa thiết kế của mình thành dạng cây của các node trong DPPT software .

2.4. Chức năng của các Menu trong thanh Menu ngang

Các hức năng trên thanh menu ngang có tác dụng đối với các ứng dụng đang chạy trên DPPT và nó cũng chứa một số lệnh đã có thao tác nhanh bằng mouse đồng thời nhiều lệnh có tổ hợp phím nóng rất tiện lợi cho các user khi làm việc và đã quen với DPPT .

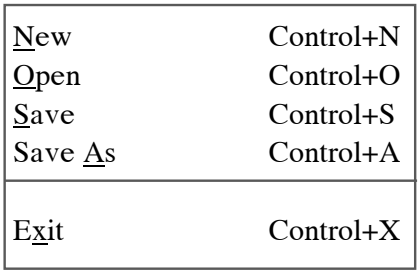
Trên Menu ngang gồm có các Menu sau :

Hình 2.4.1

Sau đây chúng ta sẽ đi vào chức năng cụ thể của từng menu :

2.4.1. File Menu

Khi user click vào mục **File** thì một sub-menu hiện ra như hình 2.4.1.1 sau :



Hình 2.4.1.1

2.4.1.1. New (Control+N)

Lệnh này cho phép xây dựng một ứng dụng mới (giải một bài toán lớn mới) . Khi click vào lệnh này (hay bấm tổ hợp phím nóng *Ctrl+N*) thì DPPT lần lượt thực hiện những công việc sau đây :

- Delete các node của ứng dụng cũ trong vùng Application .
- Khởi tạo Root node cho cây mới (xem hàm *InitNodeTree*) .
- Tiến hành detect lại các host trong mạng cũng như liệt kê ra các host đã chết kể từ detect trước .

- Khởi tạo tên ứng dụng mặc nhiên là "DPPT.DPPT" . Tên này user có thể đổi khi click vào save (cho lần *Save* đầu tiên) hoặc khi click vào *Save As* .
- Gọi chương trình *.resetdppt* để reset lại các biến toàn cục nằm tại các server của DPPT trong hệ thống mạng . Để biết về các server này độc giả hãy đọc chương 2 trong phần 3 .

Sau đây là đoạn code cho hàm new_application :

```
void new_application()
{
        delete_all_node(root_node); //delete old nodes
        InitNodeTree();

        UPDATE_NETWORK();

        save_flag=FALSE;
        My_Add_SaveAs_On();
        strcpy(Save_Filename,"DPPT.DPPT");  //re init filename

        int count=0;
        while (system(".resetdppt")<0)
        {
                cout<<"Waiting for reseting dppt system..."<<endl;
                sleep(3);
                count++;
                if (count>9)
                {
                        cout<<"Cannot reset dppt !"<<endl;
                        return;
                }
        }
}
```

2.4.1.2. Open (Control+O)

Lệnh này cho phép mở một ứng dụng đã được tạo ra và đã được *Save* hoặc *Save As* trước đó. Khi click vào mục này (hoặc bấm tổ hợp phím nóng *Ctrl+O*) thì DPPT hiển thị một cửa sổ cho phép user lựa chọn tên file của ứng dụng đã save lại trước đó . Tên mặc nhiên khi một ứng dụng được tạo mới (new) là DPPT.DPPT . Thực ra khi save một ứng dụng thì DPPT ghi các thông tin của ứng dụng vào hai file : ví dụ tên file do user nhập vào khi ghi ứng dụng là **DTK** thì DPPT sẽ tạo ra hai file có tên là **DTK** và **DTK.incl** . File *DTK* sẽ lưu lại các thông tin về các node như trạng thái máu sắc , zoom , generate , vị trí của node trong vùng Application , id của node ,... còn file *DTK.incl* sẽ lưu lại danh sách các include file của mỗi node và id của node dùng cho công việc phục hồi lại ứng dụng khi open .

Sau đây là cấu trúc dữ liệu tương ứng cho hai file được DPPT tạo ra khi save một ứng dụng (ở ví dụ trên là DTK và DTK.incl) :

```
struct SAVE_NODE
{
        char edit_file[MAX_PATHNAME];
        char in_remote_file[MAX_PATHNAME];
        char out_remote_file[MAX_PATHNAME];
        char compiler[MAX_PATHNAME];
        char xlibs[MAX_PATHNAME];
        int generate;           //TRUE or FALSE
        int x,y;                //toa do coa node
        int active;
        int zoom;
        int done;
        int parent_id;          //id cua node cha cua no
        int node_id;            //id cua chinh no
};

struct SAVE_INCLUDE_NODE
{
        char IncludeName[MAX_PATHNAME];
        int node_id;            //id cua node ma no thuoc ve
};
```

2.4.1.3. Save (Control+S)

Khi click vào lệnh này hoặc bấm tổ hợp phím nóng *Ctrl+S* thì :

- Nếu đây là lần yêu cầu thực hiện lệnh này đầu tiên thì DPPT sẽ hiển thị một cửa sổ cho user nhập vào tên của file để DPPT save trạng thái hiện thời của ứng dụng vào file đó và DPPT như đã nói cũng sẽ tự động tạo thêm một file khác (thêm đuôi là *.include*) để lưu lại tên các include file của các node trong đồ thị . Nếu user không chọn tên khác mà chỉ bấm Ok thì **hai file mặc nhiên có tên là DPPT.DPPT và DPPT.DPPT.incl** . Chú ý rằng khi mở một ứng dụng đã save trước đó thì user chỉ cần nhập vào tên DPPT.DPPT chẳng hạn còn file DPPT.DPPT.incl thì DPPT software sẽ tự động tìm và mở để đọc các thông tin đã lưu trước đó . Nếu thiếu một trong hai file này hoặc có lỗi trong quá trình mở file thì DPPT sẽ thông báo lên shell mà DPPT đang chạy .
- Nếu đây không phải là lần đầu user yêu cầu ghi lại *ứng dụng này* thì DPPT tự động lấy tên của lần trước mà user nhập vào để tiến hành ghi lại trạng thái của ứng dụng (tất nhiên với hai file) .

Một chú ý rất quan trọng nữa là user phải tự điều khiển lấy công việc lưu lại các ứng dụng của mình , DPPT sẽ không "nhắc nhở" các user nếu như họ không save lại ứng dụng của mình và lại thoát DPPT hay mở một ứng dụng mới (!!!) .

2.4.1.4. Save As (Control+A)

Lệnh này có chức năng tương tự như Save nhưng cho phép user ghi lại ứng dụng với một tên mới được nhập vào từ cửa sổ nhập của window Save As mà DPPT sẽ hiển thị cho user khi họ click vào lệnh này hoặc bấm tổ hợp phím nóng Ctrl+A .

2.4.1.5. Exit (Control+X)

Khi bấm Ctrl+X hoặc click vào lệnh này thì DPPT sẽ thoát và giải phóng luôn các sever của nó trên các host trong hệ thống mạng . Cần chú ý rằng nếu user không dùng lệnh này mà lại dùng lệnh quit của hệ thống trên menu ở góc trên bên trái của cửa sổ ứng dụng DPPT hoặc dùng lệnh kill của Unix thì các server trên mạng sẽ không được DPPT giải phóng . Nếu gặp trường hợp thứ hai thì sau đó tại command line user hãy đánh vào *.shutdowndppt* và enter . Chương trình .shutdowndppt sẽ giải phóng các server của DPPT trên toàn mạng (tất nhiên chỉ các server trong phiên làm việc này của DPPT chứ không phải của các lần làm việc trước còn sót lại chưa được giải phóng , đó là lỗi của user và họ phải tự kill các server đó) .

Mỗi khi lệnh này được yêu cầu thực thi thì DPPT gọi hàm Exit_Proc . Sau đây là source code cho hàm này , mời các độc giả tham khảo nhanh :

```
void
Exit_Proc (Widget w, XtPointer client_data, XtPointer xt_call_data )
{
        XmPushButtonCallbackStruct *call_data = (XmPushButtonCallbackStruct *) xt_call_data;

        //********************
        int count=0;
        while (system(".shutdowndppt")<0)
        {
                cout<<"Exiting ..."<<endl;
                sleep(3);
                count++;
                if (count>9)
                {
                        cout<<"Cannot shut down dppt system !"<<endl;
                        exit(-1);
                }
        }
        exit(0);
        //********************
}
```

2.4.2. View Menu

Khi user click vào mục **View** thì một sub-menu hiện ra như hình 2.4.2.1 sau :

Global Variable	Control+V
Application More	Control+M
Network Topology	Control+W

Hình 2.4.2.1

2.4.2.1. Global Variable (Control+V)

Khi click vào lệnh này hoặc bấm tổ hợp phím nóng Ctrl+V thì DPPT sẽ hiển thị một cửa sổ liệt kê tất cả các biến toàn cục trong hệ thống ở trạng thái hiện tại (các biến này nằm tại các server của DPPT) . Click mouse vào Ok để đóng cửa sổ hiển thị các biến toàn cục này lại và trở về cửa sổ hiển thị của DPPT software . Các Ok button trong các cửa sổ dạng này của DPPT đều có chức năng tương tự và sau này chúng ta sẽ không đề cập lại chức năng của chúng nữa .

2.4.2.2. Application More (Control+M)

Khi user click vào lệnh này hoặc bấm tổ hợp phím Ctrl+M thì DPPT hiển thị một cửa sổ cho toàn bộ trạng thái trong vùng Application : số node , số node tích cực ,số node zoom , số node đã được map , ... *và nếu có một node trong vùng Application được lựa chon (có màu đỏ) thì DPPT cũng cho biết node này có tích cực hay không và nếu tích cực thì đã được map hay chưa và được map lên host nào của hệ thống .* Dựa vào đặc điểm này để khi sử dụng chức năng map tự động của DPPT thì các user có thể biết được các node đã được DPPT map như thế nào ,lên những host nào của hệ thống (tất nhiên chỉ những node tích cực chưa được map thì DPPT mới map tự động còn các node khác thì nó bỏ qua , phần này sẽ được dề cập ở các mục sau) .

2.4.2.3. Network Topology (Control+W)

Khi lệnh này được thực thi thì DPPT sẽ hiển thị một cửa sổ và vẽ ra Topology của mạng các host đã được tìm thấy trong hệ thống (các host đang được hiển thị trong vùng System) . Những host nào mà không thuộc mạng hiện thời (mạng mà local host ở trên đó) thì chỉ được vẽ với một đường "gạch gạch" (dash) nối vào *bus chung* (hiện thời DPPT chỉ coi các mạng mặc nhiên có dạng bus chung) .

Công việc tìm ra được topology của mạng là một công việc rất quan trọng , từ đó chúng ta có thể đánh giá được hiệu xuất của máy , đo được tốc độ các đường truyền (tải trên các đường truyền và các máy) , ... nhằm phục vụ cho quá trình map tự động các node lên hệ thống . Đây là một ý tưởng mà DPPT hiện thời chưa thực hiện được (chức năng map tự động của DPPT hiện thời đã có nhưng còn đơn giản , chưa thể hiện được sự tối ưu) . Đây là hướng để các version sau

của DPPT cần phải thực hiện để hoàn thiện ý tưởng "dùng hệ thống phân bố để xử lý song song" .

2.4.3. System Menu

Khi user click vào mục **System** thì một sub-menu hiện ra như hình 2.4.3.1 sau :

Delete Host	Control+D
Update Host	Control+U
Update Network	Control+E
Network More Infomation	

Hình 2.4.3.1

2.4.3.1. Delete Host (Control+D)

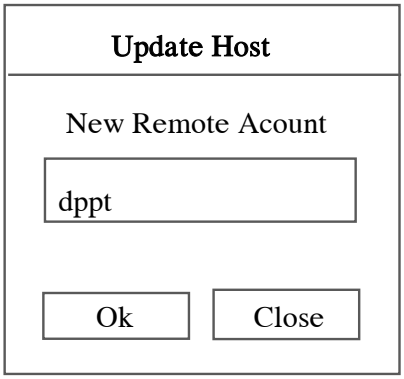
Lệnh này có chức năng tương tự như delete host bằng mouse tức là nếu host đang có node được map lên đó thì sẽ detach các node đó ra khỏi host mà không delete host này , ngược lại thì host sẽ bị xóa khởi cấu trúc dữ liệu của DPPT cũng như xóa khỏi vùng hiển thị System của hệ thống .

Muốn lệnh này được thực thi thì trước hết user phải lựa chọn host cần xóa (host sẽ chuyển sang màu đỏ) sau đó bấm Ctrl+D hoặc click vào mục này trong menu system của Menu ngang .

2.4.3.2. Update Host (Control+U)

Các host có tên trong file */etc/hosts.equiv* là các host thỏa mãn yêu cầu của DPPT tức là user có thể map các node lên đó để thực thi . Tuy nhiên các tên host trong file này còn có thể kèm theo tên của acount từ xa mà user có quyền thực thi , khi detect host thì DPPT mặc nhiên lấy username thay thế cho tên acount này và có thể không đúng như liệt kê trong file nên ở đây DPPT có chức năng này cho phép user cập nhật lại tên acount từ xa có quyền thực thi trên host này (là local host) .

Khi user yêu cầu DPPT thực thi lệnh này (click mouse hoặc bấm Ctrl+U) thì DPPT sẽ hiển thị một cửa sổ nhập như hình vẽ 2.4.3.2.1 sau đây :



Hình 2.4.3.2.1

User nhập vào tên acount của host từ xa có quyền chạy trên host này (tất nhiên phải có trong file */etc/hosts.equiv*) và bấm Ok thì host đó sẽ đổi từ màu xám sang màu xanh . Ở đây cần chú ý rằng khi cài đặt DPPT software , mỗi host trong mạng có dự định cho các node của ứng dụng trong DPPT được chạy trên đó thì trong file */etc/hosts.equiv* của host đó phải liệt kê tên của các host khác (có thể kèm theo acount hoặc không cũng được) và tương tự cho các host còn lại của mạng .

2.4.3.3. Update Network (Control+E)

Khi user yêu cầu DPPT thực hiện lệnh này thì DPPT sẽ thực hiện các công việc sau đây một cách tuần tự :

- Detach tất cả các node của ứng dụng đã được map lên hệ thống các host .
- Lưu lại danh sách (list) các host trong lần detect trước đó (đang được hiển thị trong vùng System) .
- Tiến hành detect lại các host trong hệ thống .
- So sánh hai list của các host cũ và mới để tìm ra các host đã chết so với lần detect trước .
- Vẽ lại các host trong vùng System và các node trong vùng Application .

Như chúng ta đã thấy mỗi khi detect lại các host trong hệ thống thì còn nhiều công việc liên quan nên DPPT không detect lại các host theo định kỳ mà chỉ detect lại các host khi user yêu cầu (ở lệnh này) và khi mở (open) một ứng dụng đã được save hoặc khi tao mới (new) một ứng dụng trong DPPT .

Sau đây là Source Code cho hàm UPDATE_NETWORK được goi trong ba trường hợp đã nêu trên :

```
void UPDATE_NETWORK()
{
        detach_subgraph(root_node);          //release mapped nodes
```

```
        HOST_POINTER current=head_host->next;
        while (current->next!=head_host)
                current=current->next;
        head_old_host->next=head_host->next;      //luu lai lan detect truoc
        current->next=head_old_host;
        delete head_host;

        detect_hosts();
        compare_two_list();

        int temp=0;                    //danh so lai old_host list
        current=head_old_host->next;
        while (current!=head_old_host)
        {
                current->set_host_number(++temp);
                current=current->next;
        }
        DrawHostSystem();
        DrawAllNode();
}
```

Các độc giả hãy xem thêm Source Code của hàm Open_New_Work và new_application trong phần 6 của báo cáo này để có thêm thông tin về detect host trong ba trường hợp này .

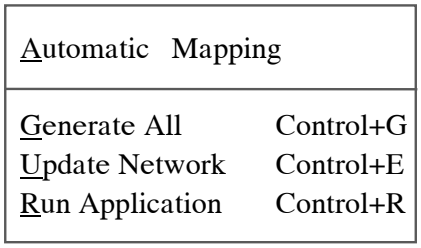
2.4.3.4. Network More Infomation

Khi user click vào mục này thì DPPT sẽ hiển thị một cửa sổ với những thông tin về trạng thái hiện tại của vùng System gồm : số host đang được hiển thị trong vùng này , thời gian kể từ lúc detect lần cuối cùng các host trong hệ thống và nếu có một host trong vùng System được lựa chọn thì cửa sổ còn hiển thị thêm thông tin về tên host , số thứ tự của host và thông tin cho biết những node nào đã được map lên host này (ở đây DPPT liệt kê id của các node được map lên host này) .

Từ đó chúng ta thấy rằng qua lệnh này chúng ta có thể biết được tình trạng của mạng cũng như của từng host trong vùng System : biết được mạng đã được detect lần cuối cách đây bao lâu để xem có cần detect lại hay không , biết được những module nào sẽ được chạy trên host màu đỏ (host đang được lựa chọn) để từ đó có những chiến lược phân bổ (map) lại các module lên các host , ... và nói tóm lại thì các lệnh trong các menu của Menu ngang đều có quan hệ mật thiết với ứng dụng của user , làm cho các user cảm thấy thuận tiện hơn khi làm việc với DPPT .

2.4.4. Implement Menu

Khi user click vào mục **Implement** thì một sub-menu hiện ra như hình 2.4.4.1 sau :



Hình 2.4.4.1

2.4.4.1. Automatic Mapping

Khi user yêu cầu DPPT thực hiện lệnh này thì DPPT sẽ tìm kiếm các node tích cực chưa được map trong vùng Application (các node có màu xanh) và phân bổ lên (map lên) các host tích cực (màu xanh hoặc tím) trong vùng System theo thuật giải sau :

- Host nào có ít node được map nhất thì DPPT sẽ map node lên host đó .
- Nếu các host có số node được map bằng nhau thì DPPT sẽ map node lên host ở về phía bên trái nhất của vùng System . Lý do cho việc này là vì khi DPPT tiến hành detect các host thì host nào tìm thấy trước sẽ được DPPT vẽ về phía bên trái của host được tìm thấy sau nó và do DPPT chưa đo được hiệu xuất của các máy nên khi map sẽ mặc nhiên coi các máy ở bên trái có tốc độ nhanh hơn (vì có response về trước khi dùng rpc broadcast tìm kiếm các host).

Để hiểu rõ hơn về phần này xin độc giả hãy tham khảo phần Source Code cho thuật giải automatic mapping sau đây :

```
void Map_by_MyAlgorithm(NODE_POINTER anode)
{
        if (anode!=NULL)
        {
                if (anode->get_active()==TRUE)
                {
                        if (anode->get_map()==FALSE)
                        {
                                int min_map=1000;    //so node maximum co the da
                                                     //map len host nao do

                                HOST_POINTER current=head_host->next;
                                HOST_POINTER map_host=current;  //host can map len

                                //****************************
                                //lay ra host duoc map it nhat
                                //****************************
```

```
                        while (current!=head_host)
                        {
                                if ((current->get_acount()==TRUE)&&
                                   (current->get_map()<min_map))
                                {
                                        min_map=current->get_map();
                                        map_host=current;
                                }

                                current=current->next;
                        }

                        //******************
                        //map on that HOST
                        //******************

                        anode->set_map(TRUE);       // trang thai map
                        anode->set_hostname(map_host->get_hostname());
                        map_host->set_map();              // map++
                }

                Map_by_MyAlgorithm(anode->peer);  //vi node ACTIVE khong
                                                  //co node con ACTIVE
        }
        else
        {
                Map_by_MyAlgorithm(anode->child);
                Map_by_MyAlgorithm(anode->peer);
        }
    }
}

void Automatic_Mapping()
{
    Map_by_MyAlgorithm(root_node);
    DrawAllNode();
    DrawHostSystem();
}
```

2.4.4.2. Generate All (Control+G)

Chúng ta đã biết chức năng generate một node trong phần **Generate Button** và ở đây chúng ta lại biết thêm về chức năng generate toàn bộ các node trong ứng dụng .

Điều kiện và các thông tin để generate một node thì chúng ta đã biết , tuy nhiên nếu trong ứng dụng có nhiều node cần generate thì việc generate từng node rất mất thời gian nên DPPT cung cấp chức năng **Generate All** này để generate tất cả các node thỏa điều kiện cần generate trong vùng Application . Muốn biết thêm thông tin về generate độc giả có thể tham khảo lại phần Generate Button (mục 2.3.6) .

2.4.4.3. Compile All (Control+P)

Cũng tương tự như trên , chúng ta đã biết về cách compile một node còn ở đây nếu user muốn compile tất cả các node tích cực của ứng dụng (bằng compiler hoặc makefile) thì họ chọn chức năng này và DPPT sẽ tự động tìm kiếm các node cần được compile (theo như DPPT qui định thì đây là các node tích cực) và gọi trình biên dịch tương ứng được chỉ ra trong property table của mỗi node đó .

Nếu quan tâm sâu về phần kỹ thuật độc giả hãy xem phần 3 của báo cáo này để hiểu cách DPPT compile một node của ứng dụng .

2.4.4.4. Run Application (Control+R)

Ngoài chức năng cho phép user có thể chạy từng node trong một ứng dụng thì DPPT còn cung cấp chức năng cho phép user chạy toàn bộ các node tích cực tức là chạy toàn bộ ứng dụng khi user click vào mục này hoặc bấm tổ hợp phím nóng Ctrl+R . Đây cũng là bước để các user xem được kết quả cuối cùng của ứng dụng được thiết kế trên DPPT .

Thêm một thông tin có thể có ích cho user là DPPT sẽ chạy lần lượt các node tích cực của ứng dụng *từ trái sang phải* . Độc giả có thể đọc phần 3 của báo cáo đề tài này để biết kỹ thuật mà DPPT sử dụng để thực thi (run) một node của ứng dụng .

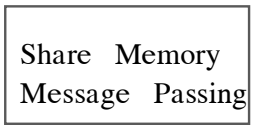
Sau đây là đoạn code cho chức năng này của DPPT để độc giả có thể tham khảo nhanh :

```
void Pre_Run_Application(NODE_POINTER anode)
{
	if (anode!=NULL)
	{
		if (anode->get_active()==TRUE)
		{
			Run_Node(anode);
			Pre_Run_Application(anode->peer); //vi node ACTIVE khong
                                                  //co node con ACTIVE
		}
		else
		{
			Pre_Run_Application(anode->child);
			Pre_Run_Application(anode->peer);
		}
	}
}
```

```
void Run_Application()
{
        Pre_Run_Application(root_node);
}
```

2.4.5. Options Menu

Khi user click vào mục **Options** thì một sub-menu hiện ra như hình 2.4.5.1 sau :



Hình 2.4.5.1

2.4.5.1. Share Memory

Mục này và mục sau (2.4.5.2) chỉ đại diện cho những chức năng mở file đơn giản của DPPT với trình soạn thảo *textedit* của Unix nhưng nó lại có ý nghĩa quan trọng khi user hiểu mình đang mở những file gì của DPPT software . Thậy vậy , hai mục này cho phép user mở ra hai file cấu hình của DPPT đại diện cho mô hình *Share Memory Sever* hoặc mô hình *Message Passing* : DPPT hỗ trợ cho cả hai mô hình này . Hai file này chỉ ra đường dẫn cho các chương trình server cần chạy . *Để hiểu rõ về thiết kế cũng như thực thi phần hệ thống này của DPPT software , độc giả hãy xem chương 2 của phần 3 và phần 4 của báo cáo này .*

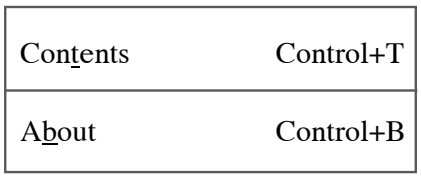
Mục này của DPPT sẽ mở file config : **DPPT_SHARESERVER.CONFIG** đại diện cho mô hình Share Memory Server .

2.4.5.2. Message Passing

Với lệnh này DPPT sẽ mở file config : **DPPT_MSGPASS.CONFIG** đại diện cho mô hình Message Passing .

2.4.6. Help Menu

Khi user click vào mục **Help** ở tận cùng bên phải của menu ngang thì một sub-menu hiện ra như hình 2.4.6.1 sau :



Hình 2.4.6.1

2.4.6.1. Contents (Control+T)

Ngoài chức năng quick help giới thiệu nhanh chức năng của các button trong DPPT thì DPPT còn cung cấp chức năng help cho các user để học cách sử dụng phần mềm DPPT .

Khi click vào mục này hoặc bấm tổ hợp phím nóng Ctrl+T thì DPPT hiển thị một window với những thông tin cần thiết cho user về cách sử dụng phần mềm DPPT . Tuy vậy chức năng help này của DPPT còn khá đơn giản , chỉ là load một file help và hiển thị nội dung của nó lên màn hình . Tương lai cần phát triển sao cho DPPT có chức năng help on-line .

2.4.6.2. About (Control+B)

Chức năng này hiển thị thông tin về các tác giả cũng như một vài thông tin khác có liên quan đến đề tài này . Bấm vào mục *"Click Here !"* để đóng cửa sổ About .

2.5. Tóm tắt qui trình thao tác với DPPT của các user

Đến thời điểm này các user đã biết cách sử dụng phần GUI của DPPT , tuy nhiên để có một cái nhìn khái quát hơn về GUI thì trong mục này chúng tôi sẽ giới thiệu tóm tắt lại qui trình thao tác trên DPPT . Khi thao tác với DPPT thì thông thường các user phải thực hiện các bước sau đây:

1. User phải thiết kế (design) ứng dụng (bài toán) của mình : phân chia ứng dụng thành bao nhiêu module và thể hiện thiết kế của mình trên vùng Application của DPPT với mỗi node của vùng này đại diện cho một module của ứng dụng . Cần chú ý rằng các module có thể được phân chia thành nhiều module nhỏ hơn và DPPT cung cấp cấu trúc các node có dạng cây (tree) để thể hiện được điều này nhằm tạo ra sự tương đồng giữa thiết kế của của user và mô hình của bài toán được thể hiện trong DPPT .
2. User nhập vào các properties của mỗi node tích cực trên đồ thị vùng Application (thường là các node lá ngoại trừ một trường hợp node có thể là node trung gian nhưng các node con cháu của nó phải ở trạng thái zoom thì node trung gian này mới có ý nghĩa theo qui ước của DPPT trong version hiện thời : DPPT Version 1.0) .
3. Soạn thảo cho các node tích cực của đồ thị ứng dụng nhờ Source Code button như đã trình bày trong các mục trước .
4. User tiến hành Generate cho các node trong đồ thị ứng dụng (nếu có node dạng như vậy) . Có thể generate cho từng node hay cho cả đò thị và DPPT tự tìm ra các node cần generate .
5. User dùng các chức năng của DPPT có thể là tự động hay thủ công để map các node của ứng dụng lên các host tích cực trong vùng System . Có thể user cũng cần phải update lại một host nào đó nếu như quá trình detect host của DPPT chưa nhận dạng được host đó là tích cực nhưng user lại biết chắc điều đó qua thông tin trong file /etc/hosts.equiv .

6. User tiến hành compile các node ứng dụng : có thể compile từng node hay compile cả đồ thị ứng dụng (khi đó DPPT sẽ tự động tìm kiếm các node cần compile) .
7. Tiến hành chạy các module của ứng dụng trên các host mà chúng được map lên sau khi đã compile thành công node ứng với module đó . Có thể chạy từng node hoặc chạy cả ứng dụng đồng thời .

Trong các bước ở trên thì bước 2,3,4 và 5 có thể thực hiện xen kẽ lẫn nhau không quan tâm đến thứ tự thực hiện của chúng

Chú ý :

Phần tiếp theo là phần 3 - thiết kế và thực thi phần hệ thống của đề tài . Nếu không muốn đi sâu vào phần hệ thống của DPPT mà chỉ cần sử dụng phần mềm DPPT thì độc giả có thể bỏ qua phần tiếp theo và đọc phần 4 - thư viện cho người lập trình .

PHẦN 3

THIẾT KẾ VÀ THỰC THI PHẦN HỆ THỐNG CỦA ĐỀ TÀI

CHƯƠNG 1

THIẾT KẾ VÀ THỰC THI PHẦN HỆ THỐNG CỦA GIAO DIỆN VỚI NGƯỜI SỬ DỤNG

CHƯƠNG 2

PHÂN TÍCH , THIẾT KẾ CÁC CHƯƠNG TRÌNH HỆ THỐNG VÀ THƯ VIỆN LẬP TRÌNH CHO NGƯỜI SỬ DỤNG TOOL

CHƯƠNG 1

THIẾT KẾ VÀ THỰC THI PHẦN HỆ THỐNG CỦA GIAO DIỆN VỚI NGƯỜI SỬ DỤNG

1. Giới thiệu chung

DPPT Version 1.0 là một phần mềm dùng làm công cụ hỗ trợ cho các programmer trong khi lập trình để xử lý song song trên mạng phân bố tức là dùng mạng phân bố để giải quyết các bài toán song song thay vì dùng một máy tính có nhiều CPU với giá thành quá cao nên không phù hợp với túi tiền của những người sử dụng máy tính bình thường .

Với ý tưởng " dùng mạng phân bố để xử lý song song " như vậy , vấn đề đặt ra là phải thiết kế DPPT sao cho vừa dễ sử dụng nhưng cũng phải mô hình hóa được các bài toán theo đúng ý tưởng thiết kế của user để họ không cảm thấy bỡ ngỡ khi chuyển bài toán của mình từ bản thiết kế vào DPPT .

Để đạt được yêu cầu đó , các tác giả của DPPT đã quyết định thiết kế DPPT thành hai phần chính là : phần giao tiếp với các user và phần hệ thống mức thấp hơn như đã nói trong *phần 1* của báo cáo này . Phần giao tiếp với các user được thiết kế gồm một *giao diện đồ họa* thân thiện với user để họ có thể mô hình hóa bài toán của mình trong DPPT một cách dễ dàng và một *thư viện lập trình* với nhiều hàm có chức năng mạnh để user có thể thực hiện trao đổi dữ liệu giữa các node trong ứng dụng của mình . Phần hệ thống mức thấp của DPPT có nhiệm vụ thực hiện các chức năng mà user yêu cầu trong phần trên , đây là một phần dùng để giao tiếp giữa DPPT và hệ thống Unix cũng như tạo ra cầu nối giữa các node , các host của ứng dụng được thể hiện trong dạng graphic với các cấu trúc dữ liệu của chúng trong DPPT nhưng các user không cần phải quan tâm tới phần này mà chỉ tập trung vào phần trên để giải quyết triệt để bài toán của họ .

Với thiết kế như vậy , trong bản báo cáo này ở phần 3 chúng tôi sẽ giới thiệu các vấn đề có liên quan sâu đến kỹ thuật thực thi bên dưới của DPPT mà cụ thể là trong chương này các độc giả sẽ biết được cách DPPT tìm kiếm các host , quản lý các module , cách DPPT compile một node , ... còn ở chương sau độc giả sẽ được giới thiệu về các kỹ thuật mà DPPT sử dụng để thực hiện việc trao đổi dữ liệu giữa các node của DPPT từ đó đưa ra các hàm thư viện lập trình cho user ở phần 4 của báo cáo này . Sau đây mời các độc giả đi vào phần chi tiết về kỹ thuật của từng phần trong thiết kế trên .

2. Quản lý các host trong hệ thống của DPPT

Đứng về phía các user thì họ chỉ thấy các host được hiển thị trong vùng System của DPPT với những đặc tính , trạng thái khác nhau cùng một vài thông tin như hostname , địa chỉ host và số module được map lên host nhưng user không thấy được cấu trúc dữ liệu bên trong của DPPT để quản lý các tính chất đó của host , như không biết DPPT làm thế nào để tìm được các host trong mạng , ... Trong phần này chúng tôi sẽ giới thiệu về các đặc tính kỹ thuật đó của DPPT software .

2.1. Cấu trúc dữ liệu cho các host

Vì DPPT *được viết bằng C++* nên để tận dụng sức mạnh của ngôn ngữ thì tất cả các cấu trúc dữ liệu cho host hay cho node đều được thiết kế thành các Object tức là các class trong ngôn ngữ C++ .

Sau đây là cấu trúc dữ liệu để quản lý các host trong hệ thống của DPPT software :

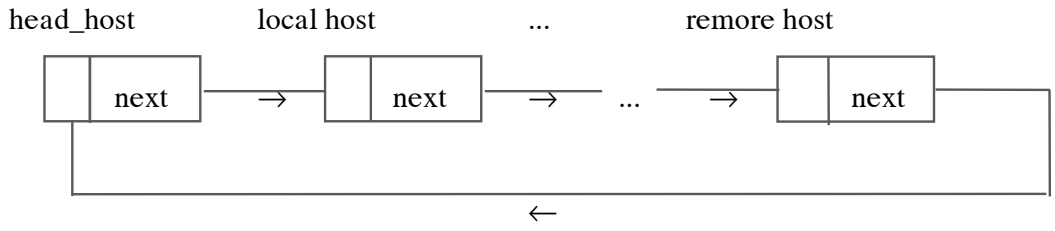
```
#define MAX_NAME 20
#define MAX_ID 20

class HOST
{
        public:
                HOST():acount(FALSE),map(0),host_number(0),selected(FALSE)
                {
                        strcpy(hostname,"unknown");
                        strcpy(id,"unknown");
                        strcpy(name_acount,"unknown");
                }; //constructor

                void set_acount(int a);
                void set_map();       //tang len 1
                void decrease_map();  //giam di 1
                void both_map(int m);  //+1 hoac -1
                void set_host_number(int h);
                void set_selected(int s);
                void set_hostname(char h[MAX_NAME]);
                void set_id(char i[MAX_ID]);
                void set_name_acount(char n[MAX_ID]);

                int get_acount();
                int get_map();
                int get_host_number();
                int get_selected();
                char* get_hostname();
                char* get_id();
                char* get_name_acount();

                ~HOST() {};

                HOST *next;             //tro den host ke tiep trong list

        private:
                int acount;             //TRUE,FALSE - co hay khong co acount
```

```
        int map;                //so module da map len host nay
        int host_number;        //so thu tu cua host
        int selected;           //host duoc lua chon khong ?
        char hostname[MAX_NAME];  //ten host
        char id[MAX_ID];        //dia chi internet
        char name_acount[MAX_ID]; //ten acount cua may tu xa
};

typedef HOST* HOST_POINTER;

HOST_POINTER head_host,head_old_host;
```

Trong cấu trúc dữ liệu của host chúng ta thấy có 7 biến riêng (private) là :

- *acount :* nếu DPPT có quyền chạy một chương trình trên host này thì biến acount=TRUE, ngược lại acount=FALSE . *Chú ý rằng local host luôn được DPPT hiển thị tại tận cùng bên trái của vùng System và biến acount của nó luôn luôn bằng TRUE .*
- *map :* cho biết có bao nhiêu node đã được map lên host này .
- *host_number :* số thứ tự của host khi DPPT tìm thấy nó và đồng thời cũng là số được hiển thị trong vùng System tại mỗi host .
- *selected :* TRUE nếu host đang được lựa chọn (host này sẽ có màu đỏ) . Trong vùng System tại một thời điểm chỉ có một host có biến selected=TRUE .
- *hostname :* lưu lại tên host với tối đa là 20 ký tự .
- *id :* lưu địa chỉ của node dạng "dot" .
- *name_acount :* lưu lại tên acount của host mà DPPT có quyền chạy chương trình trên acount đó của host này.

Ngoài ra chúng ta còn phải chú ý tới *con trỏ next* trong class HOST được trỏ tới host kế tiếp trong host list . Các host được lưu trữ theo dạng danh sách liên kết vòng nên con trỏ next của host cuối cùng trong list sẽ trỏ về con trỏ *head_host* . Con trỏ *head_old_host* lưu lại list của các host đã chết so với lần detect trước đó . Để cho dễ hình dung chúng ta xem hình vẽ minh họa cách lưu trữ các host 2.1.1 sau đây :



Hình 2.1.1

Chú ý rằng head_host chỉ là một con trỏ dạng HOST_POINTER dùng để đánh dấu host list chứ thực ra không lưu trữ dữ liệu về các host trong hệ thống . Sau đây chúng ta sẽ xem xét cách mà DPPT version 1.0 tìm ra các host trong mạng Unix .

2.2. Kỹ thuật tìm kiếm các host trong mạng

DPPT sử dụng broadcast để tìm các dịch vụ RPC của Unix trên các host mà cụ thể ở đây DPPT sử dụng dịch vụ *rpcbind* . Dịch vụ này được khởi tạo mỗi khi một host khởi động để vào hệ điều hành Unix . Để tìm kiếm các host theo kỹ thuật này chúng ta có thể dùng hai hàm RPC do Unix cung cấp là rpc_broadcast hoặc rpc_broadcast_exp . Hai hàm này có nhiều tham số cần truyền , tuy nhiên để tìm được các host thì chúng ta chỉ quan tâm đến hai tham số của hai hàm này là *const u_long prognum , const u_long versnum* : prognum là số program number của dịch vụ RPC còn versnum là số version của dịch vụ RPC (ở đây là rpcbind) . Prototype của hai hàm RPC này như sau :

```
enum clnt_stat rpc_broadcast(const u_long prognum,
        const u_long versnum, const u_long procnum,
        const xdrproc_t inproc, const caddr_t in,
        const xdrproc_t outproc, caddr_t out,
            const resultproc_t eachresult, const char *nettype);

enum clnt_stat rpc_broadcast_exp(const u_long prognum,
        const u_long versnum, const u_long procnum,
        const xdrproc_t xargs, caddr_t argsp,
        const xdrproc_t xresults, caddr_t resultsp,
        const resultproc_t eachresult, const int inittime,
        const int waittime, const char *nettype);
```

Cả hai hàm này có chức năng tương tự nhau ngoại trừ hàm rpc_broadcast_exp có khởi tạo thêm thời gian timeout : inittime là thời gian mà rpc_broadcast_exp chờ trước khi gửi lại yêu cầu một lần nữa , sau lần gửi đầu tiên thì thời gian này được tăng lên theo cấp số mũ cho đến khi vượt quá waittime thì rpc_broadcast_exp chấm dứt gửi các yêu cầu . Chú ý rằng inittime và waittime được tính theo milliseconds (một phần nghìn giây) . Hàm rpc_broadcast sẽ sử dụng thời gian timeout với giá trị default thường là 20 giây .

Trong khi tìm kiếm các host thì DPPT sử dụng hàm rpc_broadcast_exp với inittime=2 và waittime=512 .

Sau đây là chi tiết của các hàm liên quan đến kỹ thuật detect host của DPPT , có thể có những include file hay các biến không được dùng cho những hàm này (dùng cho các hàm khác của DPPT nhưng cũng được khai báo ở đây) thì độc giả không cần quan tâm . *Chú ý rằng phải có include file host.h thì các thủ tục này mới thực hiện được vì còn nhiều include file đã được khai báo trong file host.h :*

```
#include <Xm/Text.h>
#include<time.h>
```

```
#define RPC_NAME_DEFAULT "rpcbind"    //ten dich vu rpc dung detect host
#define MAX_VERSION 4                 //max version cua dich vu rpc

extern long start_time;
extern int b;                         //boolean variable

int host_number=0,delete_host_flag=FALSE;

void detect_hosts()
{
        init_hostlist();    //khoi tao danh sach host
        detect_hostlist();  //tim cac host (co va khong co acount),tao hostlist
        time(&start_time);
}

void init_hostlist()
{
        head_host=new HOST();
        HOST_POINTER localhost=new HOST();    //host cuc bo-day la SPARCSERVER20
        struct hostent *hp;
        host_number=0;                  //so host

        char name[MAX_NAME];
        sysinfo(SI_HOSTNAME,name,MAX_NAME);
        localhost->set_hostname(name);

        hp=gethostbyname(name);
        struct in_addr* ptr=(struct in_addr*) *(hp->h_addr_list);
        strcpy(name,inet_ntoa(*ptr));      //name==id
        localhost->set_id(name);          //IP of host

        struct passwd *pw ;
        pw =getpwuid(getuid());
        localhost->set_acount(TRUE);
        localhost->set_name_acount(pw->pw_name);   //acount of user

        localhost->set_host_number(++host_number);  //set so thu tu cua host
        localhost->set_selected(TRUE);

        localhost->next=head_host;
        head_host->next=localhost;
}

int findhost(char* namehost)
{
```

```
        HOST_POINTER current=head_host->next;
        while
         ((current!=head_host)&&(strcmp(current->get_hostname(),namehost)!=0))
                current=current->next;
        if (current!=head_host)
                return 0;      //da co roi
        else    return 1;      //chua co
}

void delete_ahost(char* namehost)    //theo ten
{
        HOST_POINTER current=head_old_host->next,before=head_old_host;
        while
           ((current!=head_old_host)&&
            (strcmp(current->get_hostname(),namehost)!=0))
        {
                before=current;
                current=current->next;
        }
        if (current!=head_old_host)
        {
                before->next=current->next;
                delete current;
        }
}
void delete_all_old_host()     //xoa cac host trong old list of host
{
        char name[MAX_NAME];
        HOST_POINTER current=head_old_host->next;

        while (current!=head_old_host)
        {
                strcpy(name,current->get_hostname());
                current=current->next;
                delete_ahost(name);
        }
}

void add_ahost(HOST_POINTER ahost)
{
        int b=findhost(ahost->get_hostname());
        if (b==1)         //chua co host nay
        {
                ahost->set_host_number(++host_number); //them vao so thu tu
```

```
                HOST_POINTER current=head_host;
                while (current->next!=head_host)
                        current=current->next;  //ve cuoi danh sach
                ahost->next=head_host;
                current->next=ahost;            //them vao cuoi danh sach
        }

}

void who_alive(caddr_t out,struct netbuf *addr,struct netconfig *netconf)
{
        char *host1,host[MAX_ID];
        int i=0,j=0,len;
        u_long addr1;
        struct hostent *hp;
        host1=taddr2uaddr(netconf,addr);
        if (host1!=NULL)
        {
                len=strlen(host1);
                for (i=0;i<len;i++)
                {
                        if (host1[i]=='.') j++;
                        if (j==4) break;
                        host[i]=host1[i];
                }
                host[i]='\0';
                if ((int)(addr1=inet_addr(host)) == -1)
                        cout<<"Error: Dia chi phai la dang a.b.c.d"<<endl;
                else
                {
                        hp=gethostbyaddr((char*)&addr1,sizeof(addr1),AF_INET);
                        if (hp!=NULL)
                        {
                                HOST_POINTER newH=new HOST();

                                newH->set_hostname(hp->h_name);
                                newH->set_id(host);

                                struct passwd *pw ;
                                pw =getpwuid(getuid());
                            int kt;
                                kt=ruserok(hp->h_name,0,pw->pw_name,pw->pw_name);
                            if (kt==0)
                                        {
                                        newH->set_acount(TRUE);
```

```
                                newH->set_name_acount(pw->pw_name);
                                }
                        else
                                newH->set_acount(FALSE);
                        add_ahost(newH);
                }
        }
}

void detect_hostlist()
{
        enum clnt_stat rpc_stat;
        struct rpcent *my=getrpcbyname(RPC_NAME_DEFAULT);
        for (int ver=1;ver<MAX_VERSION;ver++)  //tim tat ca cac version san co
        {
                rpc_stat=
            rpc_broadcast_exp(my->r_number,ver,NULLPROC,xdr_void,(caddr_t)NULL,
                xdr_void,(caddr_t)NULL,(resultproc_t)who_alive,2,256,(char*)NULL);
                if (rpc_stat!=RPC_SUCCESS)
                        if (rpc_stat!=RPC_TIMEDOUT)
                        {
                                cout<<"Error: "<<clnt_sperrno(rpc_stat)<<endl;
                                exit(0);
                        }
        }  //for
}
```

2.3. Các vấn đề liên quan khác

Sau khi tìm ra các host trong hệ thống thì vấn đề là hiển thị các host cùng các trạng thái của chúng,cập nhật thông tin cho các host trong khi user thao tác với DPPT,xóa các host, ... Đó là những công việc mà DPPT đã thực hiện được và Source Code của các chức năng này đếu được liệt kê đầy đủ trong phần 6 của báo cáo và ở trong các source file : *host_tools.c , dppt_dppt.c và node_tools.c* .

3. Quản lý các node trong hệ thống của DPPT

Như đã nối trước đây , các module của ứng dụng được mô hình hóa thành các node trong DPPT và các node này được lưu trữ dưới dạng cây để gần với mô hình thực tế khi thiết kế bài toán . Cấu trúc dữ liệu và các thao tác trên các node của DPPT là một phần rất phức tạp với nhiều biến , nhiều hàm chức năng . Sau đây chúng ta sẽ đi vào xem xét cụ thể cấu trúc dữ liệu cũng như các hàm thao tác trên cấu trúc dữ liệu đó của các node trong DPPT software .

3.1. Các cấu trúc dữ liệu

Các cấu trúc dữ liệu có liên quan đến node được liệt kê trong file **node.h** , ở đây chúng tôi chỉ đưa ra cấu trúc dữ liệu để lưu các include file của node (class INCLUDE) và cấu trúc dữ liệu của node (class NODE) để các gới thiệu với các độc giả . Các hằng số dùng trong hai cấu trúc dữ liệu này đều được khai báo trong file node.h (độc giả có thể xem ở phần 6) .

Cấu trúc dữ liệu để lưu trữ các include file của một node :

```
class INCLUDE
{
        public:
                INCLUDE():next(NULL) { strcpy(include,"nothing"); };
                INCLUDE(char i[MAX_PATHNAME]):next(NULL) {strcpy(include,i);}

                void set(char i[MAX_PATHNAME]);
                char* get();

                INCLUDE* next;

                ~INCLUDE() {};

        private:
                char include[MAX_PATHNAME];
};

typedef INCLUDE* INCLUDE_POINTER;
```

Cấu trúc dữ liệu cho node :

```
typedef int int_xy[2];

class NODE
{
        public:
                NODE():x(0),y(0),generate(FALSE),active(TRUE),map(FALSE),id(0),
                    selected(FALSE),parent(NULL),child(NULL),peer(NULL),
                    zoom(FALSE),done(FALSE)
                {
                    strcpy(edit_file,DEFAULT_EDIT_FILENAME);
                    strcpy(in_remote_file,DEFAULT_REMOTE_IN_FILE);
                    strcpy(out_remote_file,DEFAULT_REMOTE_OUT_FILE);
                    strcpy(hostname,NODE_DEFAULT_HOSTNAME);
                    strcpy(xlibs,XLIBS_DEFAULT);
                    strcpy(compiler,DEFAULT_REMOTE_COMPILER);
```

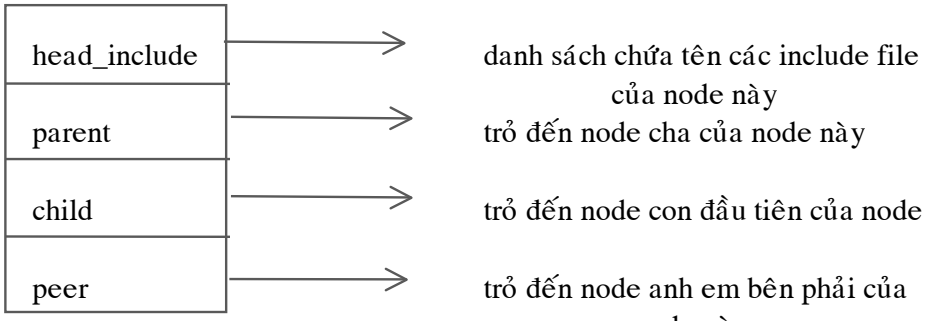
```
        //khoi tao include list
        head_include=new INCLUDE();
        head_include->next=head_include;  //ring linked-list
};

void set_edit_file(char edit[MAX_PATHNAME]);
void set_in_remote_file(char r_in[MAX_PATHNAME]);
void set_out_remote_file(char r_out[MAX_PATHNAME]);
void set_xlibs(char x[MAX_PATHNAME]);
void set_compiler(char c[MAX_PATHNAME]);
void set_generate(int g);
void set_x(int xset);
void set_y(int xset);
void set_xy(int xset,int yset);   //both of them
void set_map(int m);
void set_active(int a);
void set_zoom(int z);
void set_selected(int s);
void set_hostname(char h[MAX_NAME]);
void set_id(int i);
void set_done(int d);

void set_include(char i[MAX_PATHNAME]);
void delete_include(char i[MAX_PATHNAME]);
void show_include(Widget w);

int get_done();
int get_id();
int get_selected();
int get_active();
int get_map();
int get_zoom();
int get_x();
int get_y();
void get_xy(int_xy *xy);       //xy[0]==x , xy[1]==y
int get_generate();
char* get_out_remote_file();
char* get_in_remote_file();
char* get_edit_file();
char* get_hostname();
char* get_xlibs();
char* get_compiler();
```

```
        INCLUDE_POINTER head_include;   //chua include file list

        NODE* parent;
        NODE* child;
        NODE* peer;

        ~NODE()   //destructor
        {
            //xoa head_include list

            INCLUDE_POINTER current=head_include->next;
            INCLUDE_POINTER before=head_include;
            while (current!=head_include)
            {
                before->next=current->next;
                delete current;
                current=before->next;
            }
            delete head_include;
        };

    private:
        char edit_file[MAX_PATHNAME];
        char in_remote_file[MAX_PATHNAME];
        char out_remote_file[MAX_PATHNAME];
        char hostname[MAX_NAME];
        char xlibs[MAX_PATHNAME];
        char compiler[MAX_PATHNAME];
        int  generate;      //TRUE or FALSE
        int  x,y;           //cot,hang ma node duoc ve
        int  map;           //TRUE or FALSE
        int  active;        //TRUE or FALSE
        int  selected;      //TRUE or FALSE
        int  zoom;          //TRUE or FALSE
        int  done;          //TRUE or FALSE : da generate chua
        int  id;            // so de nhan dang node
};

typedef NODE *NODE_POINTER;

NODE_POINTER root_node;
```

Vì cấu trúc dữ liệu của node được thể hiện rất rõ ràng trong phần khai báo trên nên ở đây chúng tôi không nói thêm về chức năng của các biến trong hai cấu trúc dữ liệu trên mà chỉ đưa ra hình vẽ minh họa cho cấu trúc của một node (hình 3.1.1) và hình vẽ minh họa cho

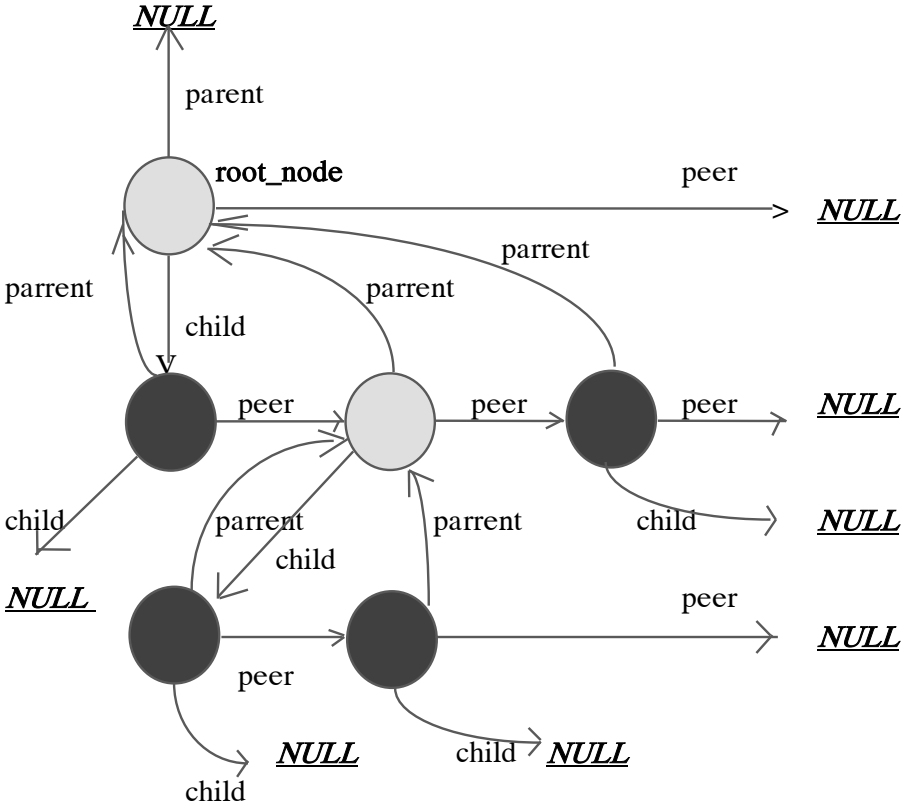
cấu trúc bên trong của một đò thị ứng dụng trong DPPT (hình 3.1.2) . Tuy nhiên cũng nên chú ý rằng mỗi node có 4 con trỏ : parent trỏ tới node cha của nó để tiện lợi cho quá trình zoom in , zoom out và nhiều chức năng khác nữa ; peer trỏ tới node anh em của node này (node con cuối cùng của một node nào đó sẽ có con trỏ peer=NULL) ; child trỏ tới node con đầu tiên của node hiện tại (node con này có thể có nhiều node anh em cũng như có thể có các node con cháu khác nữa) ; cuối cùng là con trỏ head_include trỏ tới danh sách các include file của node này (đây cũng là một danh sách liên kết vòng : *ring linked-list*) .

Cấu trúc một node :



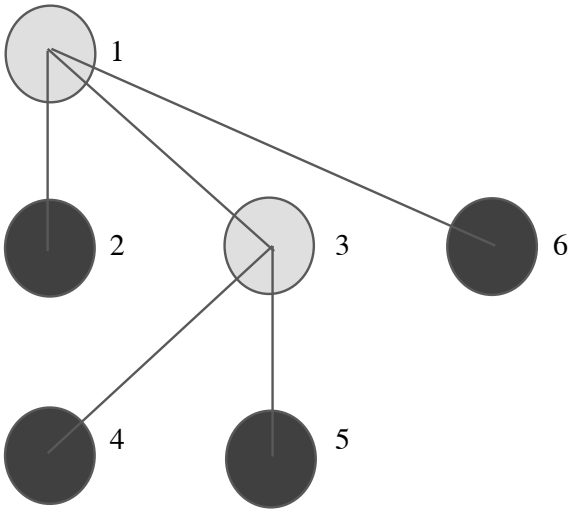
Hình 3.1.1

Minh họa cấu trúc bên trong của graph ứng dụng :



Hình 3.1.2

Cấu trúc cây ứng dụng ở hình 3.1.2 sẽ được hiển thị trong vùng Application của DPPT như hình 3.1.3 sau :



Hình 3.1.3

3.2. Các vấn đề liên quan

Xung quanh cấu trúc của node là một loạt các hàm chức năng trong file DPPT_DPPT.c có thao tác với các trường trong cấu trúc dữ liệu của node cũng như trong các cấu trúc dữ liệu có liên quan được liệt kê trong file *node.h* và *host.h* . Các hàm này tạo nên nền tảng cho phần GUI của DPPT , nó thực hiện các chức năng liên quan đến các node trong đồ thị của vùng Application cũng như các host trong vùng System và một vài chức năng trong Menu ngang (như Save , Save As , Open , Generate , ...) . Các độc giả hãy xem phần 6 của báo cáo để biết thêm thông tin về các hàm chức năng này .

4. Kỹ thuật Compile một node của DPPT

Như chúng ta đã được biết , DPPT là một phần mềm hỗ trợ cho xử lý song song trên mạng phân bố và nó có thể chạy trong một mạng phân bố không đồng nhất (homogeneous distributed network) . Để làm được điều này thì điều quan trọng nhất là làm sao DPPT tạo ra được các chương trình thực thi chạy được trên các homogeneous computer ?

Chúng ta biết rằng các máy tính thuộc các chủng loại khác nhau trong mạng phân bố chưa chắc chạy được các chương trình do các máy khác trên mạng tạo ra (có thể không tương thích về bộ lệnh , về cách lưu trữ các dữ liệu (số integer có thể 2 hay 4 byte chẳng hạn) , ...), chính vì điều này nên DPPT Version 1.0 đã đưa ra cách giải quyết đơn giản là chương trình do máy nào tạo ra (mỗi chương trình là một node) thì chỉ máy đó chạy . Muốn đem một chương trình (còn gọi là một node , một module) sang máy khác chạy thì máy khác đó phải tự biên dịch lại Source Code (tất nhiên với sự hỗ trợ của DPPT) . Cần lưu ý độc giả rằng phần 5 của báo cáo này

có đề ra các hướng phát triển của đề tài và cũng đưa ra một hướng mới để phát triển cho phần này nhằm giảm bớt thời gian cho quá trình compile này .

Với ý tưởng biên dịch (compile) như vậy của DPPT thì mỗi khi user yêu cầu DPPT dịch một node nào đó (*đã được map* lên một host từ xa) bằng *Test Button* hay chức năng trên menu ngang *Implement/Compile All*, DPPT sẽ chuyển cả Source Code và các include file của node đó sang máy từ xa (thư mục để chuyển sang ở máy từ xa đã được chỉ ra trong Node Properties) và gọi C (hay C++) Compiler đã có sẵn trên máy từ xa để dịch chương trình ngay trên máy từ xa đó . Cách thực hiện này của DPPT đảm bảo rằng một node khi đã được map lên một máy trong hệ thống thì chương trình thực thi cuối cùng do DPPT tạo ra cho node sẽ hoàn toàn tương thích với máy mà node được map lên .

Trong phần này , DPPT có đưa ra một kỹ thuật nhằm tạo ra khả năng *multiuser* cho DPPT . Chúng ta biết rằng thư mục đích để đưa Source Code đến ở máy từ xa cũng có thể được *một user khác cũng đang sử dụng DPPT* chọn làm thư mục đích cho chương trình của họ và một vấn đề là làm sao DPPT tránh cho các file của hai user này không được trùng tên (vì các user có thể hoàn toàn không biết cách đặt tên file source của người kia) . Để làm được điều này DPPT thêm vào tên file Source Code của các node một chuỗi gồm: HOST ID, GROUP ID , USER ID và PROCESS ID của bản DPPT mà họ đang chạy . Chú ý rằng ở đây tôi phải thêm vào PROCESS ID của bản DPPT user đang chạy vì một lý do đã gặp tromh thực tế khi thực thi DPPT : nhiều user cùng login vào một server , trong một acount chung và họ cùng sử dụng DPPT (tất nhiên mỗi user chạy một bản DPPT riêng) và điều này làm cho khả năng mà các user đặt tên file trùng nhau là có thể . Tuy nhiên khi tôi thêm PROCESS ID vào tên file thì sự việc lại hoàn toàn khác hẳn (!!!) . Các phần thêm khác vào tên file là dùng cho các user khác nhau trên mạng cùng chạy DPPT .

Một chú ý quan trọng ở đây là cả các include file cũng được thêm vào tên một chuỗi như trên do đó user phải khai báo các include do họ tạo ra trong **#Include Button** chứ không được thêm trực tiếp vào trong Source Code chương trình của các node (nếu có) .

Sau đây là tổng kết lại các bước mà DPPT compile một node trong đồ thị ứng dụng :

- *Nếu node được map lên một remote host :*

1. Thêm vào tên các include file (nếu có) của node một chuỗi gồm 4 mục như đã nói ở trên và copy các file đó tới remote host . Chú ý rằng các file trong khi DPPT thực thi đều là file tạm và khi user view Source Code hay xem các include file thì tên file hoàn toàn không có gì thay đổi (có tính transparent đối với các user) .
2. Sinh thêm các dòng *#include <new filename of include files>* vào trong Source Code chương trình của node .
3. Sinh thêm các biến hệ thống : được sử dụng nếu node có liên lạc với các node khác trong ứng dụng .
4. Thêm vào tên file Source Code chương trình của node một chuỗi như đã thêm ở bước 1 (nếu bước 1 có thể không được thực thi thì bước này là bắt buộc với các node trong trường hợp này) và sau đó copy Source Code file này sang remote host tới thư mục được chỉ ra trong Node Properties (các include file được mặc nhiên chép tới cùng thư mục với Source Code file) và với tên mới (nếu user muốn) cũng được chỉ ra trong Node Properties .

5. Gọi C (hay C++) compiler (được chỉ ra trong Node Properties) để biên dịch chương trình này thành file thực thi với tên cũng được chỉ ra trong Node Properties .
6. Nếu biên dịch thành công thì DPPT báo : Compiled Node <id> . Ngược lại thì lỗi sẽ được hiển thị lên shell mà bản DPPT này đang được chạy .

- *Nếu node được map lên local host :*

Tương tự như trên tuy nhiên bước 1 và bước 4 được bỏ qua .

Ở đây cần nói thêm rằng DPPT còn hỗ trợ cả makefile , lúc đó nếu node được map lên host từ xa thì DPPT chép cả thư mục (hoặc các file) được chỉ ra trong *local edit pathname* (có thể dưới dạng các ký tự đại diện) của Node Properties tới host từ xa (không có đổi tên hay làm thêm thao tác nào cả) vào thư mục được chỉ ra trong *remote input pathname* và gọi chương trình make trên host từ xa từ thư mục này . Nếu node được map lên local host thì DPPT gọi make program với file "(M)makefile" nằm trong thư mục được chỉ ra trong *remote input pathname* . Sau đây là đoạn source code của DPPT hỗ trợ cho makefile :

....

```
//*******************************
//compile by makefile
//*******************************

sprintf(str,"%s",anode->get_compiler());
if (strcmp(str,"make")==0)
{
        if (strcmp(hostname,head_host->next->get_hostname())==0)
        {
                sprintf(str,"cd %s ; make",in_remote_file);

                count=0;
                while (system(str)<0)
                {
                        cout<<endl<<"Cannot Compile by make program : ";
                        cout<<anode->get_id()<<"...Waitting..."<<endl;
                        sleep(3);
                        count++;
                        if (count>9)
                        {
                                cout<<endl<<"Tried over,bye..bye !"<<endl;
                                exit(-1);
                        }
                }
        }
        else
```

```
        {
            sprintf(str,"rcp %s %s:%s",anode >get_edit_file() , hostname ,
                in_remote_file);

            count=0;
            while (system(str)<0)
            {
                cout<<endl<<"Cannot Compile by make program for";
                cout<<"Node : "<<anode->get_id()<<"(rcp) !"<<endl;
                cout<<"Waitting..."<<endl;
                sleep(3);
                count++;
                if (count>9)
                {
                    cout<<endl<<"Tried over,bye..bye !"<<endl;
                    exit(-1);
                }
            }

            sprintf(str,"rsh %s \"cd %s ; make\"",hostname,in_remote_file);

            count=0;
            while (system(str)<0)
            {
                cout<<endl<<"Cannot Compile by make program for";
                cout<<"Node : "<<anode->get_id()<<"(rsh) !"<<endl;
                cout<<"Waitting..."<<endl;
                sleep(3);
                count++;
                if (count>9)
                {
                    cout<<endl<<"Tried over,bye..bye !"<<endl;
                    exit(-1);
                }
            }
        }

        cout <<endl<<"Compiled Node : "<<anode->get_id()<<endl;
        return;
    }

....
```

5. Kỹ thuật Run một node của DPPT

Để chạy một chương trình trên host từ xa , DPPT sẽ dùng lệnh **rsh** của Unix . Khi user yêu cầu DPPT thực thi một node (nếu thực thi cả ứng dụng thì DPPT cũng lần lượt chạy từng *node tích cực* từ trái sang phải) thì nếu node được map lên remote host , DPPT sẽ dùng rsh để chạy chương trình trên remote host đó , ngược lại DPPT chạy chương trình bình thường như khi user đánh lệnh vào từ command tool .

Cấu trúc của *rsh command* cũng như các lệnh khác của unix các độc giả có thể tham khảo cuốn *using unix* được liệt kê trong phần tài liệu tham khảo (sau cùng của báo cáo này).

Sau đây là Source Code của thủ tục run một node của DPPT : Run_Node

```
void Run_Node(NODE_POINTER anode)
{
	int child,count;
	char str[2*MAX_PATHNAME],hostname[MAX_PATHNAME];

	system(".runtimereset");

	char host_portserver[20],port_portserver[20];
	getportserver(host_portserver,port_portserver);

	if (strcmp(anode->get_out_remote_file(),"")==0)
	{
		cout<<"Nothing for Node "<<anode->get_id()<<" to Run"<<endl;
		return;
	}

	strcpy(hostname,anode->get_hostname());
	if (strcmp(hostname,NODE_DEFAULT_HOSTNAME)==0)
		strcpy(hostname,head_host->next->get_hostname());

	sprintf(str,"%s",anode->get_out_remote_file());

	//*************************tao ra filename duy nhat*****************
	if (strcmp(hostname,head_host->next->get_hostname())!=0)
	{
		struct passwd *pw ;
		pw =getpwuid(getuid());

		char append[MAX_PATHNAME];
		sprintf(append,".%s.%d.%d.%d.cc",head_host->next->get_id(),
			pw->pw_gid,pw->pw_uid,getpid());

		strcat(str,append);
```

```
        }

        //*********************************************************************

        child=-1;count=0;
        while ((child=fork())<0)
        {
                cout<<endl<<"Cannot fork process to run Node ";
                cout<<anode->get_id()<<endl;
                cout<<"Waitting..."<<endl;
                sleep(3);
                count++;
                if (count>9)
                {
                        cout<<"Co gang qua 9 lan...bye..bye !"<<endl;
                        exit(-1);
                }
        }

        if (child==0)        //child process
        {
              if (strcmp(hostname,head_host->next->get_hostname())==0)
                        if (execlp(str,str,host_portserver,port_portserver,NULL)<0)
                        {
                                cout<<"Cannot Run "<<str<<endl;
                                exit(-1);
                        }
                        else    {}
                else
                {
                        if  (execlp("rsh","rsh",hostname,str,host_portserver, port_portserver ,
                                    NULL)<0)
                        {
                                cout<<"Cannot Run "<<str<<endl;
                                exit(-1);
                        }
                        else    {}
                }
        }
}
```

Các thủ tục khác có liên quan đến vấn đề run một node hay run một ứng dụng của DPPT các độc giả có thể tham khảo trong Source Code file : *DPPT_DPPT.c* ở phần 6 của báo cáo.

CHƯƠNG 2

PHÂN TÍCH , THIẾT KẾ CÁC CHƯƠNG TRÌNH HỆ THỐNG VÀ THƯ VIỆN LẬP TRÌNH CHO NGƯỜI SỬ DỤNG TOOL

NỘI DUNG :

- ◊ A. PHÂN TÍCH CÁC YÊU CẦU
- ◊ B. CÁC CÔNG CỤ LẬP TRÌNH ĐƯỢC HỆ THỐNG HỖ TRỢ
- ◊ C. THIẾT KẾ VÀ HIỆN THỰC CHƯƠNG TRÌNH

A. PHÂN TÍCH CÁC YÊU CẦU

1.Vấn đề trao đổi dữ liệu giữa các process trên hệ thống:

- Khi user dùng graphic editor tạo ra một đồ thị ,mỗi node chứa một đoạn chương trình của user sẽ được thực thi trên một host nào đó do user chỉ định .Khi thực thi mỗi node là một user task . Giữa các user task có sự trao đổi dữ liệu nhau. Khi chúng ta map các node lên các host và thực thi thì chúng có thể trao đổi dữ liệu với nhau .Các user task được thực thi trên các host khác nhau trong hệ thống .Tất nhiên là không phải khi nào tất cả các host trên hệ thống này cũng đồng nhất nhau . Khi thực thi mỗi user task là một process riêng biệt .Để đảm bảo cho việc trao đổi dữ liệu giữa các task đó chúng ta phải xem xét các vấn đề sau đây :

- Mô hình trao đổi dữ liệu dữ liệu dùng chung cho các node trong hệ thống mà có nhu cầu dùng chung data,trao đổi data và các vấn đề liên quan .
- Vấn đề interprocess communications giữa các process trên hệ thống .
- Vấn đề data representation .

1.1 Mô hình trao đổi dư liệu dùng chung :

- Tài nguyên dùng chung nói chung hay các biến toàn cục nói riêng được tất cả các node trong hệ thống xử dụng .Do đó để thao tác trên dữ liệu dùng chung chúng ta cần xem xét những khía cạnh sau đây :

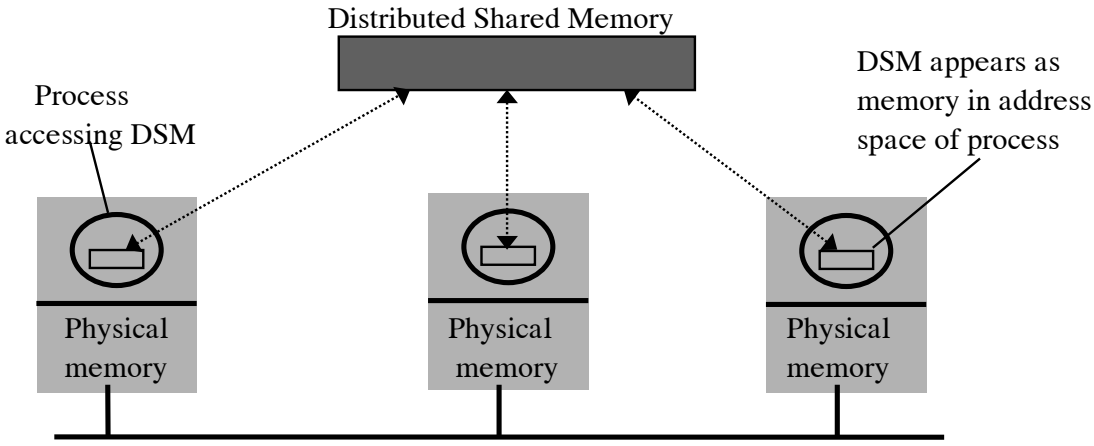
- Exchange Model .
- Data synchronization .
- Concurrency control .
- Transaction

1.1.1 Mô hình trao đổi :

- Để trao đổi tài nguyên dùng chung trên hệ thống chúng ta có hai mô hình chính sau :

1.1.1.1 Distributed Share Memory :

- Distributed Share Memory (DSM) là một khái niệm trừu tượng được sử dụng cho việc chia xẻ data giữa các máy tính mà không thể chia xẻ bộ nhớ vật lý .Các processes truy xuất DSM bằng các tác vụ read và update giống như bộ nhớ bình thường trong không gian địa chỉ của nó .Tuy nhiên có một hệ thống run-time bên dưới đảm bảo các process đang thực thi tại các computer khác nhau nhận biết được sự thay đổi trên DSM .Mặt dù rằng process truy xuất một shared memory đơn nhưng thực ra thì bộ nhớ vật lý là phân tán .
- Hệ thống DSM run-time hỗ trợ cho việc gởi các message để update trong hệ thống cũng như quản lý các data trùng lặp :từng máy có một bản sao các dữ liệu được truy xuất trong DSM vì lý do tốc độ truy xuất .



The distributed shared memory abstraction

Các khía cạnh liên quan đến vấn đề hiện thực DSM :

◊ **Hardware_base :** Một vài kiến trúc multiprocessor dựa trên phần cứng đặt biệt để xử lý vấn đề load và store các lệnh áp dụng ở các địa chỉ trên DSM ,và dùng truyền thông với các remote memory module khi cần thiết để hiện thực . Khi thiết kế các processor và memory được kết nối nhau thông qua một high_speed network .

◊ **Page-based :** một số như Ivy ,Munin ,Mirage ,Clouds ,Mether ... tất cả hiện thực DSM như là một vùng nhớ ảo có cùng mức địa chỉ trong không gian địa chỉ của từng process thành viên . Trong từng trường hợp kernel sẽ bảo đảm tính consistency dữ liệu trong vùng DSM như là một phần của việc sử lý page fault .

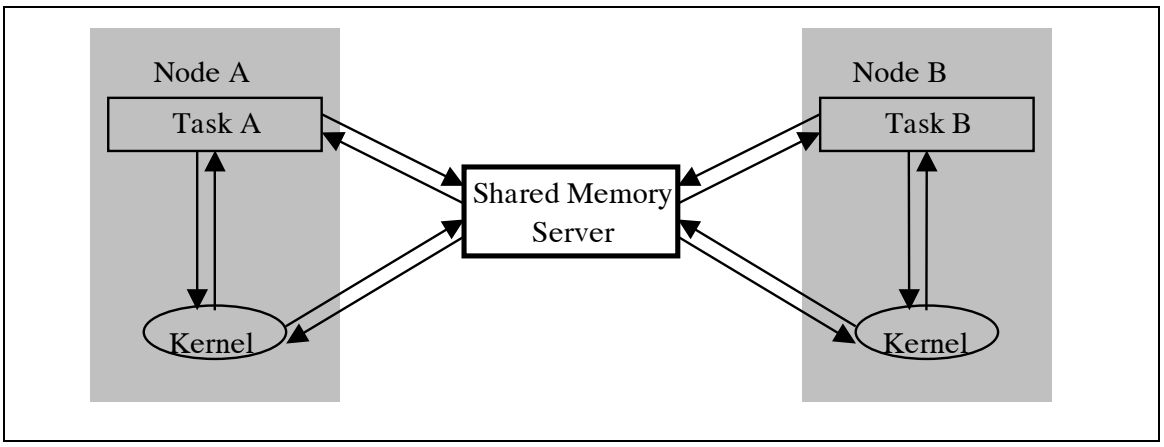
◊ **Library-based :** Một số ngôn ngữ hoặc mở rộng ngôn ngữ như Orca và Linda hỗ trợ các dạng DSM . Trong kiểu này hiện thực việc chia sẻ không thực hiện qua hệ thống bộ nhớ ảo mà bằng truyền thông giữa các instance của language run-time .Các process tạo các lệnh gọi thư viện được đưa vào bởi compiler khi nó truy xuất trong DSM . Các thư viện truy xuất các thành phần local và liên lạc khi cần thiết để đảm bảo tính consistency .Các process có thể thực thi các thread để xử lý các messages đến có các yêu cầu thao tác trên DSM .

- Khi thiết kế DSM chúng ta cần phải lưu ý các vấn đề sau :
  - Cấu trúc dữ liệu lưu trên DSM .
  - Mô hình đồng bộ sử dụng truy xuất DSM .
  - Tính consistency của mô hình DSM .
  - ...

Network Shared Memory Server :

- Server là một user level task cho phép các client trên mạng truy xuất dữ liệu dùng chung .
  - Ưu điểm : dữ liệu có một bản ,dễ cập nhập ,dễ thấy tất cả client, thiết kế đơn giản, dễ đồng bộ . Đó là những mặt mạnh so với DSM trên các máy khác nhau .
  - Khuyết điểm: tạo phần tử yếu ,vai trò của Server rất quan trọng .Có nhiều yêu cầu truy xuất nên dễ tạo bottleneck .

- Tuy nhiên đối với hệ thống mạng của chúng ta và những công cụ mà chúng ta hiện có mô hình này rõ ràng cũng có nhiều phù hợp hơn với công cụ của chúng ta hơn so với mô hình DSM mà mỗi máy có một bản lưu Shared Memory .



1.1.1.2 Message Passing :

- Trong mô hình này tại một host có một chương trình daemon .Chương trình này có nhiệm vụ truyền các message . Do đó những task ở các host khác nhau có thể thông qua daemon này mà truyền các message trên tới các task khác một các dễ dàng như tại local .Trong mô hình này vấn đền IPC sử dụng hai tác vụ căn bản là send và receive .
- Một message trước khi gởi đi phải được xây dựng trước tại không gian của sending process .
- Một ứng dụng dùng message passing theo nhiều cách :
  - Gởi message mà không mong đợi reply .
  - Chờ một message đến và xử lý nó .
  - Gởi message mong đợi reply nhưng không chờ .
  - Gởi message và mong reply .

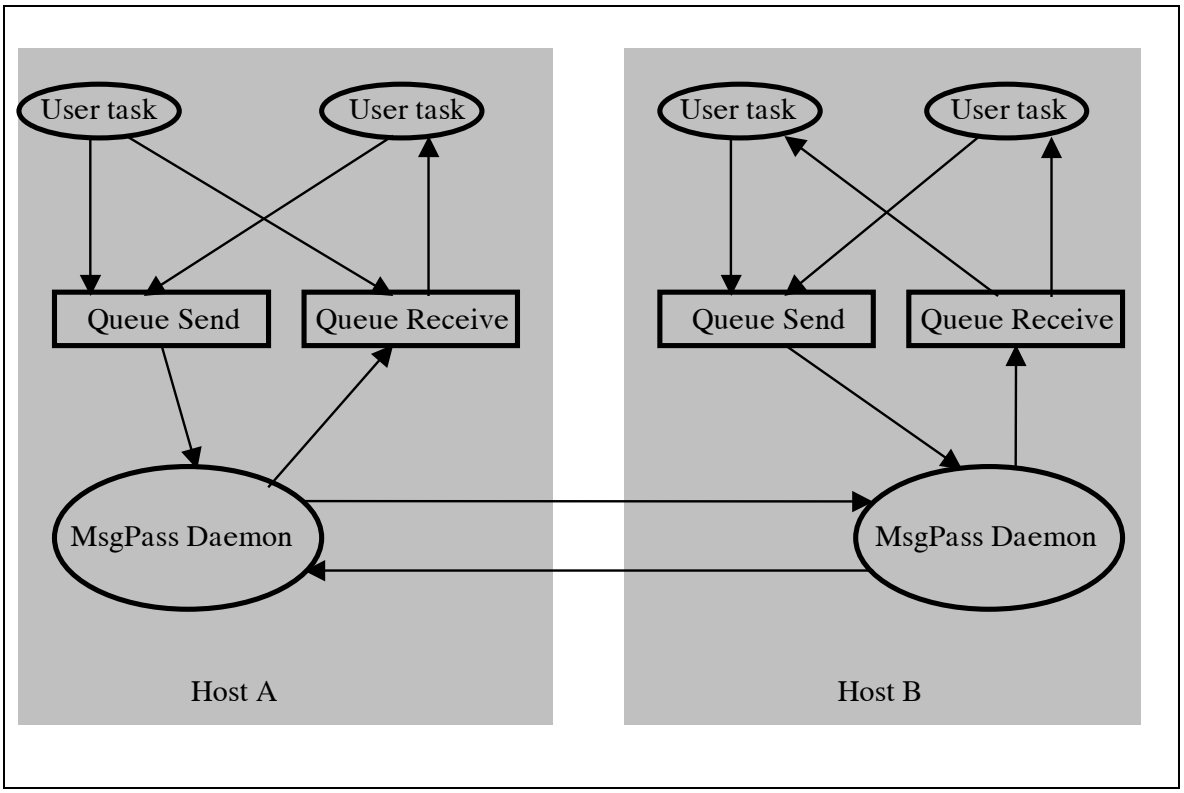
Design Issues for Message Passing :

-Khi thiết kế Message Passing System thì chúng ta cần phải quan tâm đến các vấn đề sau :

- ◊ send / receive message : bao gồm việc mất message ,truyền lại ...
- ◊ naming : một vấn đề rất lớn đó là làm thế nào để xác định được tên của process nhận và process gởi . Nếu như hệ thống mạng lớn và không có một central manager quản lý tên thì có thể dẫn đến trùng tên .
- ◊ Authentication :làm thể nào để xác nhận đối tượng mà trao đổi là đúng đắn ...

tất nhiên là còn nhiều vấn đề nữa nhưng trong công cụ của chúng ta nếu sử dụng mô hình message passing thì chúng ta cần quan tâm đến các vấn đề truyền nhận và định danh .

- Giống như mô hình DSM message passing cũng không tránh khỏi một số vấn đề trong hệ thống phân bố .



1.1.1.3 Message Passing versus DSM :

- Một số vấn đề giữa DSM và message passing sau đây :

◊ Programming model : trong mô hình message passing biến được marshall từ một process ,truyền và unmarshall vào một biến khác trong process nhận .So với shared memory một biến được nhận trực tiếp cho nên không cần quá trình marshalling . Một đặc tính nữa là việc định danh và truy xuất một biến qua DSM như là một biến không share .Trong khi đó trong message passing mỗi process được bảo vệ vì có một không gian địa chỉ riêng còn DSM thì một process có thể làm cho một process bị fail .Nếu như message passing khi truyền giữa các máy tính khác loại với nhau (heterogeneous computer thì quá trình marshalling đảm bảo được sự biểu diễn data còn DSM thì không đảm bảo được .Việc đồng bộ giữa các process trong message passing model thông qua các tác vụ primitive của message passing trong khi đó thì DSM có thể được thực hiện qua lock hay semaphore . Process trong DSM có thể tạo được persitent ,các process có thể thực thi với thời gian sống non-overlapping .Process có thể để data cho các process khác thao tác sau khi kết thúc .Đối với message passing thì giữa hai process phải đồng thời .

◊ Efficiency : trong một vài chương trình thì xây dựng các chương trình parallel theo DSM và message passing trên hệ thống phấn bố có hiệu quả như nhau trên cùng một hardware .Tuy nhiên thực tế thì không phải lúc nào cũng vậy .

-Có một vấn đề khác biệt về cost liên quan đến hai kiểu lập trình . Trong message passing tất cả dữ liệu truy xuất điều rõ ràng do đó người lập trình phải lưu ý khi có những thao tác hay xử lý mà liên quan đến communication . Trong khi đó sử dụng DSM bất kỳ một tác vụ đọc ghi nào cũng có thể có hoặc không liên quan đến communication bằng các hỗ trợ lớp dưới của hệ thống .

1.1.1.4 Chọn lựa mô hình thích hợp cho công cụ :

- Đối với mô hình DSM thì rõ ràng đây là một mô hình rất hiệu quả nếu chúng ta có một lớp dưới đảm bảo vấn đề transparency và consistency .Tuy có hạn chế về số lượng các máy tham

gia do vấn đề bottleneck nhưng với thực tế của chúng ta thì nếu ta sử dụng DSM theo thông thường nghĩa là mỗi máy có một bản lưu DSM giống nhau ,mà trong khi hệ thống của chúng ta không có cung cụ hỗ trợ lúc bấy giờ công việc của chúng ta là rất nhiều và phức tạp .Đồng thời khi xây dựng lớp dưới đảm bảo cho vấn đề transpanrency và consistency của vùng DSM thì rõ ràng là vấn đề truyền nhận giữa các máy để đảm bảo hai đặt tính trên là rất lớn .Mỗi tác động lên một vùng DSM ở một máy nào đó thì lập tức hệ thống phải đảm bảo tác động đó phải xảy ra ở tất các các DSM còn lại và chúng ta lại dễ dàng dẫn đến vấn đề bottleneck .

- Cả hai mô hình trên tùy theo từng bài toán cũng như trong từng hệ thống mà có ưu điểm cũng như khuyết điểm .Nếu chúng ta thiết kết một công cụ mà quá phụ thuộc vào hệ thống chỉ riêng của chúng ta thì điều đó không tốt cũng như đứng trên quan điểm của một người thiết kế công cụ thì mục tiêu của chúng ta sẽ là làm sao cho càng có nhiều chức năng cũng như thích ứng dễ dàng với nhiều môi trường và thân thiện với user thì càng tốt .Đứng trên quan điểm đó nên trong quá trình thiết kế công cụ này chúng ta sẽ xây dựng cả hai mô hình .Đối với DSM chúng ta sử dụng mô hình Network Shared Memory Server với nhiều server .

1.1.2 Data Synchronization :

- Việc đồng bộ có hai vấn đề :
  - đồng bộ cho hệ thống khi có nhiều process của nhiều client cùng truy xuất dữ liệu dùng chung .
  - đồng bộ của chương trình user.

1.1.2.1 Đồng bộ trong hệ thống :

- Để thực hiện đồng bộ cho hệ thống khi có nhiều client cùng truy xuất dữ liệu dùng chung thì ta có một số cách :
  - Clock Synchronization .
  - Mutual Exclusion .
  - Election algorithms.

với các giải thuật như :

+Distributed algorithm useful logical clock

+Berkeley algorithm

+Ring-based algorithm

+Discussion of distributed mutual exclusion algorithm

...

-Trong các cách trên thì Mutual Exclusion là thích hợp với chương trình của chúng ta .Sử dụng Mutual Exclusion ta có một số kiểu biến đồng bộ như sau :

- Mutex Exclusion Lock
- Multiple -Reader ,Single -Writer Lock
- Semaphore Lock

Trong cách thông thường thì hệ thống cũng cho phép chương trình của user truy xuất dữ liệu dùng chung theo hai cách:

ME1: Safety

ME2: Liveness—dead lock free

1.1.2.2 Đồng bộ trong chương trình của User :

- Để thực hiện đồng bộ chương trình của user thì user phải sử dụng các biến ,hàm mà chương trình cung cấp trong các thư viện để tự đồng bộ chương trình của mình

1.1.3 Concurrency Program & Concurrency control :

- Vì hệ thống do nhiều task truy xuất cũng như tính thời gian của một chương đổi hỏi phải thiết kết các chương trình sao cho càng concurrency càng tốt đồng thời phải điều khiển quá trình concurrency của các process . Hệ thống của chúng ta khi xây dựng các chương trình Server chúng ta sẽ xây dựng các concurrency server . Thực hiện việc concurrency này chúng ta có thể sử dụng nhiều kỹ thuật . Ở đây chúng ta dùng kỹ thuật lập trình multithreading để hiện thực .

1.1.4 Shared data and Transactions :

- Sự giao dịch giữa client và server phải bảo đảm tính đúng đắn của dữ liệu .Một giao dịch hoặc thành công hoặc bị loại bỏ bởi client hay server .Các vấn đề mà chúng ta cần quan tâm trong sự giao dịch giữa client và server :

- hành động của dịch vụ khi hư hỏng .
- hành động của client liên quan đến sự hư hỏng của server .
- vấn đề mất dữ liệu update .
- Inconsistent retrievals
- Serial equivalence .

...

1.2 Interprocess Communication:

- Việc truyền thông giữa các process trên hệ phân bố ta xem xét hai trường hợp sau :

- IPC giữa các process trên một host .
- IPC giữa các process ở các host khác nhau .

1.2.1 IPC giữa các process trên cùng một host :

- Vấn đề IPC giữa các process trên cùng một host là một vấn đề đã được giải quyết rất nhiều .Hệ thống UNIX cũng cung cấp rất nhiều các hàm hệ thống về phần này .

1.2.2 IPC giữa các process ở các host khác nhau :

-Để trao đổi dữ liệu dùng chung giữa các process ở các host khác nhau thì có nhiều cách:

- Client -Sever model .
- RPC model
- Các mô hình khác .

-Trong bài toán của chúng ta sử dụng mô hình Client-Server .Khi thao tác trên dữ liệu chung ở NSM thì chúng ta sử dụng cơ chế client/server .Còn đối với mô hìng Message Passing thì các client khác nhau sẽ trao đổi dữ liệu dùng chung qua trung gian là là một server chạy background truyền nhận message .

1.2.3 Communication :

a) Clients and Server :

-Chương trình client -server có thể dựa trên hai giao thức sau :

- Connection -oriented protocol như TCP/ IP :cung cấp độ tin cậy khi truyền ,đảm bảo dữ liệu đến theo đúng thứ tự .
- Conectionless request / reply protocol như UDP/ IP : ưu điểm là đơn giản ,không cung cấp độ tin cậy .

-Khi cần truy xuất dữ liệu chung các clients gởi yêu cầu tới Server và Server sẽ thực thi theo cầu . Ở đây có nhiều các sử dụng message passing như sau :

- gởi message không mong đợi reply .
- chờ một message tới và xử lý chúng .
- gởi message trong đợi reply và không đợi nó khi nào tới thì sẽ xử lý sau .
- gởi message và chờ reply .

-Trong công cụ của chúng ta khi truyền thông qua hệ thống mạng thì chúng ta sử dụng giao thức TCP/ IP và gởi message có reply . Riêng đối với mô hình message passing chúng ta sẽ gởi message từ task này sang task kia thông qua daemon mà không mong đợi reply . Điều đó sẽ đúng với giả thiết rằng hệ thống của chúng ta đảm bảo . Tất cả các cầu nối thông qua mạng chúng ta điều sử dụng TCP/ IP tất nhiên là sử dụng TCP / IP sẽ hao tốn tài nguyền hơn như việc truyền nhận sẽ được đảm bảo đúng đắn .

b)Addressing:

-Để client có thể gởi các message yêu cầu thì client phải biết địa chỉ sever. Điều này được thực hiện bằng các khi một chương trình tham gia truy xuất thì hệ thống sẽ cung cấp tên cho chương trình .

- Có ba cách định địa chỉ:
    - Đưa machine number vào cố định trong client code
    - Cho proless chọn random và xác định bằng broadcast
    - Đưa sever name cho client và xác định khi thực thi

trong các cách trên thì cách thứ 3 là phù hợp với bài toán của chúng ta . Khi tool gởi chương trình của user tới remote host thì đồng thời cũng gởi luôn vị trí của server quản lý cho user task .Khi thực thi thì phần hệ thống phía user task đảm bảo được sẽ biết được vị trí của server .

c) Synchronous and asynchronous communication :

- Có hai cách khi truyền message:
    - synchronous : trong kiểu này nơi gởi và nhận điều đồng bộ nhau từng message, cả hai điều blocking .
    - asynchronous : trong kiểu này nơi gởi nonblocking còn nơi nhận thì có thể nonblocking hay blocking .

-Khi chúng ta sử dụng TCP /IP chúng ta dùng cơ chế synchronous .

d). Buffered versus unbuffered primitives

-Ở đây ta không thao tác . Thực tế khi dùng cần nối TCP/ IP thì hệ thống sử dụng buffered cho việc truyền nhận message còn ở lớp của chúng ta khi không cần buffered.

e)Reliable versus unreliable :

-Đảm bảo việc truyền message được đúng là một vấn đề quan trọng .Do vậy chúng ta phải đảm bảo rằng việc truyền phải có độ tin cậy .Vì vậy chúng ta sử dụng cơ chế báo acknowledge để đảm bảo message tới đúng .

1.3 Data representation :

- Từng máy tính có một kiến trúc có thể giống nhau hay khác nhau với các máy khách trong hệ thống phân bố .Từng kiến trúc có một kiểu biểu diễn data khác nhau .Một vài hệ thống chứa least significant byte(LSB) tại địa chỉ thấp nhất trong khi một số khác thì chứa most significant byte (MSB) tại địa chỉ thấp .Do vậy khi thực hiện các chương trình trên một hệ thống phân bố thì chúng ta phải lưu ý đến vấn đề biểu diễn data trong các hệ thống máy tính .Để có thể trao đổi data giữa các hệ thống máy khác nhau thì giữa các process gởi và nhận phải có sự thỏa thuận cách biểu diễn data .Để thực hiện điều đó ta có hai phương cách sau :
    - asymmetric data conversion : kiểu này yêu cầu một máy chuyển data của nó sang dạng biểu diễn của máy còn lại .Điều này có vấn đề là số lượng chuyển sẽ gia tăng theo số máy khác nhau trong hệ thống .Đồng thời rất khó cho người sử dụng vì đôi khi họ không biết hết kiến trúc của các máy trong hệ thống mạng .

- symmetric data conversion : sử dụng dạng biểu diễn chuẩn của mạng .Cả hai điều chuyển đổi theo dạng chuẩn và local . Sử dụng các này đổi hỏi tính toán nhiều hơn nhưng bảo đảm các process trên cùng một máy có thể sử dụng qua lại .

- Trong công cụ của chúng ta sẽ sử dụng phương pháp symmetric .
- Về việc lựa chọn dạng biểu diễn chuẩn để chuyển đổi thì có nhiều dạng chuyển đổi như ASN.1 , XDR ... công cụ chúng ta chọn XDR làm dạng chuẩn cho chuyển đổi vì đây là một dạng phổ biến và hệ thống cung cấp khá đầy đủ .

2. Vấn đề quản lý hệ thống :

2.1 Quản lý các đối tượng trong hệ thống :

- Trong quá trình phân tích các yêu cầu của tool và mô hình mà tool sử dụng thì ta thấy một vấn đề nổi lên là làm thể nào để quản lý name của các đối tượng .Tất nhiên đây là một vấn đề lớn của hệ thống phân bố .
-Name cần thiết khi một process dùng để yêu cầu các dịch vụ thực thi một tác vụ tùy theo tài nguyên và đối tượng mà dịch vụ đó quản lý .Khi một đối tượng binding một name thì lúc bấy giờ name đó sẽ đại diện cho đối tượng đó .
- Trong hệ thống của chúng ta có các đối tượng sau đây :
  -Các NDSM Server .
  -Các MsgPass Daemon .
  -Các User task .
  -Các dữ liệu dùng chung .
- Mỗi đối tượng cần có một ID để xác định cũng như mỗi đối tượng để có thể giao tiếp với đối tượng khác cần phải có một lượng thông tin nhất định .Thông tin cần thiết này bao gồm các thông tin về host ,port ,tên của đối tượng mà đối tượng đang sử dụng .Tất cả phải được quản lý vì vậy trong hệ thống cần phải có một server để thực hiện điều đó .
- Thông tin mà chúng ta quản lý sẽ bao gồm :
  -Host mà đối tượng hiện diện .
  -Port mà các đối tượng đang sử dụng để giao tiếp .
  -Tên của các đối tượng
- Trong hệ thống của chúng ta cũng không ngoại lệ ,chúng ta cần có một đối tượng quản lý name . Đối tượng đó quản lý một database về name và tất cả những đối tượng khác trên hệ thống đều biết được .Trong hệ thống của chúng ta sẽ sử dụng một server dùng để quản lý name .Chúng ta tạm gọi server đó với danh từ Port Server .Nhiệm vụ của PortServer giống như một name service Server .

2.2 Fault tolerance and recovery :

- Vì vai trò quan trọng của các server ,daemon trong chương trình cho nên vấn đề fault torelance và recovery là vấn đề cực kì quan trọng trong thiết kế tool .Đây là một vấn đề phức tạp ,khó khăn .Một trong những các giải quyết là chúng ta tạo một bản sao cho các server ,daemon .Điều này cho phép chương trình tiếp tục thực thi khi server bị hư hỏng .Trong công cụ của chúng ta sẽ giải quyết vấn đề này bằng một các đơn giản là khi một server ,hay một daemon được xác định là bị hư hỏng sau một khoản thời gian thì chúng ta sẽ cho thực thi trở lại server ,daemon đó .

2.3. Dead lock :

- Khi truy xuất dữ liệu dùng chung cũng như giao tiếp trong hệ phân bố thì khả năng deadlock luôn tồn tại .Bất kì một process nào cũng phải có khả năng ngăn cản hoặt giải quyết vấn đề deadlock .Một các giải quyết vấn đề deadlock là dùng timeout.Tất nhiên là thời gian timeout là

bao nhiêu thì rất khó xác định .Trong chương trình của chúng ta chúng ta giải quyết deadlock cho các process bằng timeout .Bất kì một tác vụ nào cũng có một khoảng thời gian timeout . Nếu một process chờ đợi một sự kiện mà quá khoảng thời gian đó thì process sẽ không chờ đợi nữa .

B. CÁC CÔNG CỤ LẬP TRÌNH ĐƯỢC HỆ THỐNG HỖ TRỢ .

- Trong các phần dưới đây chúng ta sẽ xem xet các công cụ mà ngôn ngữ lập trình cung cấp . Các công cụ này được chúng ta sử dụng để giải quyết các vấn đề có liên quan đến việc thiết kế công cụ của chúng ta .

1. Multithreaded Programming :

-Một khái niệm trừu tượng cơ bản của UNIX đó là process.Từng process xử lý các lệnh tuần tự trong một không gian địa chỉ . Hệ thống UNIX là một môi trường multiprogramming , có nhiều processes cùng hoạt động đồng thời trên hệ thống .Với những process đó hệ thống cung cấp đặc tính là máy ảo(virtual machine) .Không gian địa chỉ của process là ảo tương ứng với một vùng nhớ thực .Kernel chứa đựng nội dung của process trong nhiều đối tượng lưu trữ .
-Mỗi process có một process cha và có thể có nhiều process con .Khi hệ thống boot thì process init sẽ được tạo .

-Mô hình process có hai giới hạn quan trọng .Một là có nhiều ứng dụng muốn thực hiện nhiều tác vụ độc lập nhau mà có thể chạy đồng thời nhưng cùng chia sẻ không gian địa chỉ chung và tài nguyên điều này bắt buộc các ứng dụng phải chia ra tuần tự hay phải đưa ra một giải pháp không có hiệu quả .Thứ hai các process cũ không thể tận dụng ưu thế của multiprocessor vì một process chỉ thực thi trên một processor tại một thời điểm .Một ứng dụng có thể tạo ra nhiều process và gán chúng cho các processor còn trống .Các process phải tìm cách chia sẽ bộ nhớ và tài nguyên.

- Một process là một thực thể mà có thể xem như là có hai thành phần :
  - ◊ Tập các thread (set of threads) .
  - ◊ Tập các resources (collection of resources) .

-Thread là một đối tượng động (dynamic object) mà diễn tả một điểm điều khiển bên trong process và nó thực hiện một tuần tự các instruction .Resource được chia sẻ bởi tất cả các thread trong process . Thêm vào đó từng process có những đối tượng riêng của nó chẳng hạn như program counter ,stack ,register context .Trong quá khứ một process UNIX có là một process đơn . Multithreaded system được mở rộng cho phép nhiều thread trong một process .

-Có nhiều kiểu thread với những tính chất khác nhau
  - ◊ Kernel thread : Nó được tạo ra và hủy bởi kernel và dùng để thực thi một sốn chức năng đặt biệt .Nó chia sẽ kernel text và global data và có vùng st k kernel riêng
  - ◊ Lightweight processes(LWP) : là một thread user được hỗ trợ bởi kernel .Mọi process có thể có một hay nhiều LWP mà được hỗ trợ bởi một kernel thread riêng rẽ .LWP độc lập nhau về định thời và share các tài nguyên chung của process .Giới hạn của LWP :

- Phần lớn các hoạt động của LWP điều đòi hỏi việc gọi hệ thống (system call) điều này rất tốn kém .Vì việc gọi hệ thống đòi hỏi phải chuyển chế độ từ user mode sang chế độ sang kernel mode khi bắt đầu và ngược lại khi kết thúc .Trong mỗi lần chuyển như vậy thì LWP phải đi qua các giới hạn bảo vệ của hệ thống .Kernel chép các tham số gọi hệ thống vào kernel spac e và trả data về cho user space .

-Khi LWP thường xuyên truy xuất data thì tổng phí đồng bộ làm mất đi lợi ích hiện thực .
-Phần lớn hệ thống multiprocessor cung cấp khóa cho phép lấy data ở mức user nếu nó không bị giữ bởi những thread khác .Nếu một thread mà muốn một tài nguyên mà không có sẵn thì nó có thể thực hiện một vòng lặp cho đến khi tài nguên có mà không có sự tham gia của kernel cho nên cần phải block các thread .Block LWP đòi hỏi sự tham gia của kernel là khó thực hiện .

-Từng LWP cần có một số tài nguyên hệ thống bao gồm cả bộ nhớ vật lý cho nên hệ thống không thể cho phép nhiều LWP .Nó không thích hợp cho các ứng dụng mà dùng nhiều thread hay thường xuyên tạo và hủy thread .Đồng thời LWP cần phải định thời bởi kernel .

◊ User thread : là thread ở mức user .Kernel không biết gì user thread .Được kèm theo với các thư viện thread .các thư viện này cung cấp các hàm tạo ,đồng bộ ,quảng lý các thread mà không có hỗ trợ của kernel .Các hoạt qua lại của kernel không có liên quan đến kernel và thực hiện nhanh .Từng user thread có vùng st k user riêng là vùng dùng lưu ngữ cảnh các thanh ghi mức user ,các thông tin trạng thái khác .Thư viện sẽ định thời và chuyển đổi ngữ cảnh giữa các user thread bằng cách lưu stack và thanh ghi của thread hiện tại và load thread khác .Kernel v n giữ sự đáp ứng cho xử lý chuyển ngữ cảnh bởi vì chỉ có nó mới có quyền bổ xung cập nhập các thanh ghi quản lý bộ nhớ .User thread không thực sự là thực thể có tính định thời và kernel không biết nó .Kernel chỉ định thời các process nằm dưới và LWP sau đó những đối tượng này dùng các hàm định thời user thread .Khi process hay LWP nạp vào thì thread của nó cũng vậy .Giống như vậy khi user thread tạo một gọi hệ thống mà block (blocking system call) thì nó block LWP nằm dưới .Nếu process chỉ có một LWP thì tất cả các user thread sẽ bị block .Thư viện cũng cung cấp các đối tượng đồng bộ để bảo vệ cấu trúc data chia sẻ .Thông thường gồm các khoá và hàng các thread block trong nó .Nếu một đối tượng đã khoá ,thư viện sẽ block thread bằng các đưa nó vào hàng block .

◊ Lợi ích của user thread : sử dụng user thread có nhiều lợi ích .Nó cung cấp một cách lập trình tự nhiên nhiều ứng dụng như hệ thống windows .User thread cũng cung cấp vấn đề lập trình đồng bộ bằng cách ẩn đi các tác vụ bất đồng bộ trong thư viện các thread .Điều này làm cho chúng có ích thậm chí rằng trong các hệ thống mà kernel không hỗ trợ cho thread .Hệ thống có thể cung cấp nhiều thư viện thread mà mỗi cái tối ưu cho một lớp ứng dụng nào đó .Lợi ích lớn nhất của user thread là việc thực hiện .Việc thực hiện nó nhận kết quả từ hiện thực hàm chức năng ở mức user mà không sử dụng gọi hệ thống .Điều này tránh đi việc tổng phí cho xử lý trap và truyền các tham số ,data qua các giới hạn bảo vệ . Một lợi ích khác là critical thread size chỉ ra lượng công việc mà thread làm lợi khi như một thực thể đơn .Kích thước ở đây liên quan đến việc tạo và sử dụng thread .Với user thread thì critical size là trật tự vài trăm lệnh và khả năng giảm hơn trăm với hỗ trợ của compiler .User thread đòi hỏi thời gian ít hơn cho việc tạo , hủy và đồng bộ.

◊ Những giới hạn của User thread : bên cạnh những lợi ích user thread có nhiều giới hạn nguyên do chính là sự chia rời thông tin giữa kernel và thư viện thread .Bởi vì kernel không biết về các user thread nó không thể dùng cơ chế bảo vệ của nó để bảo vệ thread từ những cái khác .Từng process có một sở hữu không gian địa chỉ của nó mà kernel bảo vệ khỏi sự truy xuất của các process khác mà không có quyền truy xuất .User thread không có sự bảo vệ như thế, nó hoạt động trên không địa chung được sở hữu bởi process.Thư viện những thread cần phải cung cấp các yếu tố đồng bộ để thực hiện việc cùng hoạt động giữa các thread. Mô hình định thời phân chia gây ra nhiều vấn đề khác. Thư viện thread định thời user thread kernel định thời process cấp thấp hay LWP và không biết những cái khác đang hoạt động. Ví dụ như kernel có thể nạp vào một LWP có user thread đang giữ một khóa spin .Nếu một user thread trên một LWP khác trong cố gắng nhận khóa đó thì nó phải busy-wait cho đến khi đối tượng giữ chạy lại . Bởi kernel không biết mối quan hệ độ ưu tiên giữa user thread nó có thể nạp vào một LWP chạy một user thread độ ưu tiên cao định thời một LWP chạy một user thread độ ưu tiên

thấp.Cơ cấu đồng bộ mức user có thể không đúng đắn trong một số trường hợp .Phần lớn ứng dụng được viết với sự cho rằng tất cả thread chạy được là kết quả của định thời .Điều này đúng khi từng thread là giới hạn với một LWP riêng rẽ nhưng không còn đúng khi user thread được multiplex vào trong một số lượng nhỏ LWP.Bởi vì LWP có thể block trong kernel khi user thread của nó tạo mộ việc gọi hệ thống blocking một process có thể run out of (cạn kiện) LWP ngay khi có user thread và processor sẵn có . Cuối cùng không có sự hỗ trợ rõ ràng của kernel thì user thread chỉ có tể nâng cao vấn đề concurrency chứ không thể gia tăng parallelism .Thậm chí khi trên multiprocessor ,user thread chia sẻ một LWP không thể thực thi song song .

.-So với mô hình sử dụng giữa process và thread chúng ta có những nhận xét sau :

◊ Tạo mới một thread bên trong một process là rẻ hơn so với tạo một process . Chuyển đổi giữa các thread trong một process là rẻ hơn so với so với chuyển đổi giữa các thread trong các process khác nhau .

◊ Thread trong cùng process có thể chia sẻ tài nguyên một cách thuận tiện và hiệu quả hơn so với cá process khác nhau .

◊ Cùng một mức thì thread không được bảo vệ bởi các thread khác .

- Để nâng cao tính concurrency của chương trình và để có thể áp dụng chương trình trong trường hợp máy tính có nhiều processor chúng ta sẽ sử dụng công cụ thread để thực hiện các chương trình server ,daemon .

1.1 Programming with thread :

-Hiện tại các thư viện về multithreaded programming ở các hệ thống khác nhau (Solaris,Unixware) cung cấp hầu hết các hàm cần thiết cho lập trình .Các hàm trong thư viện multithreading của Solaris trên hệ thống của chúng ta như sau :

Table : Routines for Threads

Operation	Routine
Create a Thread	thr_create(3T)
Get the Thread Identifier	thr_self(3T)
Yield Thread Execution	thr_yield(3T)
Suspend or Continue Thread Execution	thr_suspend(3T) , thr_continue(3T)
Send signal to a thread	thr_kill(3T)
Access the signal Mask of the calling thread	thr_sigsetmask(3T)
Terminate a Thread	thr_exit(3T)
Wait for thread termination	thr_join(3T)
Maintain Thread-Specific Data	thr_keycreate(3T),thr_setspecific(3T) , thr_getspecific(3T)
Get The minimal Stack size	thr_min_stack(3T)
Get and Set Thread Concurrency Level	thr_getconcurrency(3T),thr_setconcurrency(3T)
Get And Set Thread Concurrency	thr_getprio(3T),thr_setprio(3T)

- Về chi tiết các hàm này xin tham khảo tài liệu Solaris Multithreaded Programming Guide của SunSoft [SOLARIS_MUL95] .

1.2 Programming with Synchronization Object :

- Trong công cụ của chúng ta để giải quyết vấn đề đồng bộ chúng ta dùng các đối tượng đồng bộ .Có nhiều đối tượng đồng bộ ở đây chỉ giới thiệu các đối tượng mà ta sử dụng trong công cụ Các đối tượng đồng bộ này là các biến đồng bộ .Trong hệ thống Solaris của chúng ta các hàm lập trình mà hệ thống cung cấp được khai báo trong <thread.h> hoặc <sync.h> và thư viện libthread.a

1.2.1 Mutual Exclusion Locks :

-Mutual exclusion locks dùng đồng bộ các threads thông thường dùng để bảo đảm rằng chỉ có một thread thực hiện đoạn code của vùng tranh chấp . Mutual exclusion lock cũng có thể dùng giải quyết tranh chấp giữa các process khác nhau . .Sau đây là các hàm của mutual exclusion lock .

Table : Routines for Mutual Exclusion Locks

Operation	Routine
Initialize a mutual exclusion Lock	mutex_init(3T)
Lock a Mutex	mutex_lock(3T)
Lock with a Nonblocking Mutex	mutex_trylock(3T)
Unlock a Mutex	mutex_unlock(3T)
Destroy Mutex state	mutex_destroy(3T)

- Về chi itết các hàm này xin tham khảo tài liệu Solaris Multithreaded Programming Guide của SunSoft [SOLARIS_MUL95] .

1.2.2 Multiple -Readers ,Single -Writer Locks :

- Reader/Writer Lock cho phép nhiều thread cùng truy xuất đọc và gới hạn cho một thread ghi . Mỗi khi một thread giữ khóa đọc thì các thread khác cũng có thể lấy khoá đọc tuy nhiên cần phải đợi để lấy khóa ghi . Nếu một thread giữ khóa ghi thì các thread khác muốn lấy khóa phải chờ đợi . Kiểu đối tượng này rất thuận lợi cho việc đồng bộ các dữ liệu được thường xuyên đọc và ít khi ghi .Các hàm của Readers/Writer Locks được trình bày sau đây :

Table : Routines for Readers /Writer Locks

Operation	Routines
Initialize a Readers/WriterLock	rwlock_init(3T)
Acquire a read lock	rw_rdlock(3T)
Try to Acquire a read lock	rw_tryrdlock(3T)
Acquire a write lock	rw_wrlock(3T)
Try to Acquire a write lock	rw_trywrlock(3T)
Unlock a Readers/Writer Lock	rw_unlock(3T)
Destroy Readers/Writer Lock State	rwlock_destroy(3T)

-Về chi itết các hàm này xin tham khảo tài liệu Solaris Multithreaded Programming Guide của SunSoft [SOLARIS_MUL95] .

2. Interprocess comunication :

- Trong một môi trường lập trình phức tạp thường xuyên nhiều process cùng hoạt động .Những process đó cần phải truyền thông với nhau ,chia sẻ tài nguyên và thông tin .Để cho phép người lập trình có thể giải quyết các vấn đề tương tác giữa các process trong hệ thống OS cung cấp

những tác vụ cho phép thực hiện điều đó ,đấy là cơ cấu *interprocess comunication* .Về tổng quan có hai mô hình cho IPC đó là :

IPC giữa các process trong cùng một hệ thống đơn (trên cùng một máy).

IPC giữa các process trên các hệ thống khác nhau .

- Đối với vấn đề IPC giữa các process trên các hệ khác nhau thì có liên quan đến networking programing .Trong phần này chúng ta chỉ đề cập đến các tiện ích cho vấn đề IPC trên hệ thống đơn .

2.1 Pipes :

- Pipe là một unstructed data stream ,đơn hướng ,FIFO (first in first out) .Data sẽ được ghi vào một cuối pipe và đọc ra ở đầu pipe .Khi đọc thì data của pipe sẽ được loại bỏ ra .Pipe cung cấp một cơ chế điều khiển luồng đơn giản . Một process đọc một pipe rỗng sẽ bị block cho đến khi có data ghi vào pipe và ngược lại một process sẽ bị block nếu cố ngắn ghi vào một pipe đầy .
- Lệnh gọi hệ thống pipe sẽ trả về hai file descriptor một cho đọc và một cho ghi . Các mô tả này được thừa hưởng bởi process con vì thế quyền chia sẻ truy xuất đến file .Như thế từng pipe có thể có nhiều process đọc và ghi .Tuy nhiên thông thường pipe được chia sẻ bởi hai process . I/O của một pipe giống như các I/O khác thông qua các hàm hệ thống read và write .

-Trong hầu hết các hệ UNIX hàm hệ thống cung cấp cho việc tạo pipe :

```
int pipe (int *filedes) ;
```

có hai file descriptors được trả về filedes[0] cho việc đọc và filedes[1] cho việc ghi .

- Trong công cụ của chúng ta không sử dụng pipe làm truyền thông giữa các process .

2.2 System V IPC :

- System V UNIX cung cấp 3 cơ chế : *semaphores ,messages queues ,shared memory* cho vấn đề IPC trên máy đơn . Về chi tiết có thể xem Interprocess Communication in UNIX [JONH 97] hoặc UNIX Network Programming [W RICHARD 90] , Unix System V/386 User‘s Guide [AT&T 88]

2.2 1 Semaphore :

- Semaphore là một đối tượng integer-valued mà có hai tác vụ nguyên thủy P() và V() . Tác vụ P() giảm giá trị của semaphore và sẽ block nếu như giá trị mới nhỏ hơn 0 .Tác vụ V() gia tăng giá trị của nó , nếu như giá trị kết quả lớn hơn hay bằng 0 thì sẽ làm cho process bị block không còn block . Semaphore được dùng trong hiện thực việc đồng bộ các đối tượng .
- Hàm hệ thống semget sẽ tạo hay nhận một dãy semaphore :

```
semid =semget(key ,count ,flag) ;
```

trong đó :

key : giá trị 32 bit do người gọi cung cấp .

- Giá trị semid dùng cho các thao tác trên semaphore . Hàm semop dùng thực hiện các thao tác trên từng semaphore trong dãy :

```
int status =semop(int semid ,struct sembuf  **sops ,unsigned int nsops) ;
```

trong đó pointer sops chỉ đến một dãy cấu trúc sau :

```
struct sembuf {
        ushort  sem_num ;  /* semaphore */
        short   sem_op ;  /* semaphore operation */
        short   sem_flg ; /* operation flag */
        } ;
```

-Hàm semctl dùng để diều khiển semaphore :

```
int semctl(int semid ,int semnum ,int cmd ,union semun arg ) ;
union semun {
        int                 val ; /* used for SETVAL only */
        struct semid_ds     *buff ; /* used for IPC_STAT and IPC_SET */
        ushort              *array ; /* used for IPC_GETALL and IPC_GETALL */
        } arg ;
```

2.2.2 Message queues :

-Message queue là một danh sách liên kết các message . Tường message chứa một giá trị ***type*** 32 bit , theo sau là data . Một process tạo hay nhận một một message queue dùng hàm hệ thống theo cú pháp sau :

```
int msgid =msgget(key ,flag) ;
```

trong đó :

key : là một số integer do user chọn .

flag : IPC_CREAT dùng tạo một message queue mới ,IPC_EXCL làm cho lệnh gọi sẽ không thành công nếu như queue đã tồn tại cho key đó .

- Giá trị msgid trả về dùng cho các lệnh gọi khác để truy xuất queue . User sẽ đặt một message vào queue bằng lệnh gọi :

```
msgsnd(msgid ,msgp ,count ,flag) ;
```

trong đó msgp chỉ đến vùng đệm message ,count là tổng số bytes trong message bao gồm cả trường type và flag chỉ đến các đọc và count trả về số bytes đọc được .

- Message được lưu giữ trong queue theo trật tự đến ,Nó được loại ra khỏi (theo FIFO) khi một process đọc :

```
count =msgrcv(msgid,msgp,maxcnt , msgtype,flag) ;
```

trong đó msgp chỉ đến vị trí mà message nhận được sẽ đặt vào maxcnt chỉ lượng data max mà có thể đọc được msgtype dùng xác định loại message và flag chỉ đến các đọc và count trả về số bytes đọc được .

- Message queue hiệu quả cho việc truyền một lượng nhỏ data nhưng rất tốn kém cho việc truyền lượng lớn data .Một đặc tính của message queue là không chỉ ra được receiver cũng như không cung cấp cơ chế broadcast .

2.2.3 Shared memory :

- Shared Memory cho phép nhiều process cùng chia sẻ vùng không gian bộ nhớ ảo . Khi một shared memory được tạo ra thì process có thể attach vùng nhớ đó và truy xuất . Shared Memory cung cấp cơ chế nhanh nhất cho các process chia sẻ data .
- Khi một process khởi tạo hay nhận một vùng shared memory thì thực hiện :

```
shmid =shmget( key , size , flag) ;
```

trong size chỉ kích thước của vùng . Sau đó process có thể attach với vùng đó qua lệnh gọi

```
addr =shmat(shmid ,shmaddr,shmflag) ;
```

shmaddr là địa chỉ vùng mà attach ,shmflag chỉ đến các cờ . Một process có thể tách (deattach) một vùng shared memory ra khỏi không gian địa chỉ của nó bằng lệnh :

```
shmdt(shmaddr);
```

- Việc hiện thực shared memory phụ thuộc nhiều vào kiến trúc virtual memory của hệ điều hành .Shared memory cho phép một lương lớn data cùng chia sẽ .Tuy nhiên hạn chế của nó là không cung cấp đồng bộ .Các process truy xuất shared memory phải tự đồng bộ .

3. Networking Interfaces :

3.1 Communication protocols :

- Dưới đây là một số protocol dùng trong vấn đề communication trên hệ thống mạng :
  - The TCP/ IP protocol suite (the Internet protocols) .
  - Xerox -Networking Systems (Xerox NS or XNS).
  - IBM's System Network Architechture (SNA).
  - IBM's NetBIOS .
  - The OSI protolcols .
  - Unix-to-Unix Copy (UUCP) .
- Trong công cụ của chúng ta sẽ sử dụng giao thức TCP/IP để truyền nhận trên hệ thống mạng . Dưới đây là một số đặt tính của giao thức TCP/IP :
  - ◊ Stream Orientation
  - ◊ Virtual Circuit Connection
  - ◊ Buffered Transfer .
  - ◊ Unstructured Stream .
  - ◊ Full Duplex Connection .

3.2 Sockets :

a. Tạo socket :

- Hàm gọi hàm hệ thống ***socket*** tạo một socket và trả về một số nguyên :

```
result =socket (af ,type ,protocol );
```

trong đó :

- af : mô tả họ protocol dùng với socket này . Các họ hiện tại bao gồm :
  - ◊ TCP /IP internet (AF_INET) .
  - ◊ Xerox Corporation PUP internet ()AF_PUP).
  - ◊ Apple Computer Incorporated Appletalk network (AF_APPLETTALK) .
  - ◊ UNIX file system (AF_UNIX).
- type : mô tả kiểu communication bao gồm :
  - ◊ reliable stream delivery service (SOCK_STREAM) .
  - ◊ connectionless datagram delivery service (SOCK_DGRAM) .
  - ◊ raw type (SOCK_RAW)
  - ◊ sequenced packet socket (SOCK_SEQPACKET) .
- protocol : thông thường được set giá trị 0 nhưng trong một số ứng dụng đặc biệt giá trị này mô tả proctocol đặc biệt .

- Giá trị trả về dùng xác định socket đó giống như file descriptor .

b . Đóng socket :

- Khi một process kết thúc việc sử dụng một socket thì nó gọi ***close*** :

```
close(socket);
```

trong đó socket mô tả cho socket đang cần đóng .

c . Specifying a local address:

- Ban đầu khi tạo socket thì chưa có liên kết nào với địa chỉ local hay đích .Chương trình server cần phải báo cho hệ thống biết nó liên kết với local address nào .

```
bind(socket,localaddr,addrlen) ;
```

d. Connecting Sockets to Destination Address :

- Trạng thái khởi tạo thì socket ở trạng thái unconnected .Lệnh connect sẽ liên kết socket với một địa chỉ đích và socket ở trạng thái connected .Trước khi truyền nhận qua socket thì phải tạo cầu nối nếu như dùng cơ chế đảm bảo .

connect(socket ,desaddr ,addrlen) ;

e. Sending &Receiving data through socket :

- Khi tạo cầu nối thì có thể sử dụng các hàm của hệ thống để truyền nhận data như :

write(socket ,buffer ,length) ;

writev(socket ,iovector,vectorlen) ;

send(socket ,message ,len,flags) ;

sendto(socket ,message,length ,flags ,desaddr,addrlen);

sendmsg(socket ,messagestruct,flags) ;

và nhận data qua socket bằng các hàm sau :

read(socket ,buffer ,length) ;

readv(socket ,iovector,vectorlen) ;

recv(socket ,message ,len,flags) ;

recvfrom(socket ,message,length ,flags ,fromaddr,addrlen);

recvmsg(socket ,messagestruct,flags) ;

f. Obtain Other Information :

- Thư viện socket cung cấp hầu hết tất cả các hàm về get và set thông tin của hệ thống mạng như thông tin về hostname ,socket ,IP address ,domain name ,protocol ...
- Chi tiết xem ở Internetworking with TCP/ IP [DOUGLAS 91] hoặc Unix Network Programming [W RICHARD 90] .

4. External Data Representation (XDR) :

- Các chương trình mà có sử dụng network-based communication giữa các máy tính luôn phải chú ý đến việc thông dịch dữ liệu truyền qua mạng .Bởi vì lý do từng máy tính có thể có kiến trúc phần cứng khác nhau rất nhiều cũng như hệ điều hành khác nhau .Do vậy chúng có thể qui ước về biểu diễn data khác nhau .Sự khác nhau đó bao gồm trật tự byte ,kích thước data ,dạng format của string và array .
- XDR standar định nghĩa kiểu biểu diễn data độc lập với máy (machine-independent) cho dữ liệu truyền qua mạng . Nó bao gồm nhiều kiểu cơ bản và các qui tắc xây dựng kiểu data phức tạp .XDR mã hoá theo từng kích thước 4 bytes
- Nếu từng máy tính mà không có kiểu biểu diễn ngữ nghĩa như XDR thì phải tốn một chi phí cho việc mã hoá ,ngoài ra vì XDR mã hoá theo 4 bytes nên trong một số trường hợp rất lãng phí .

4.1 XDR Data Types :

- Dưới đây là một số kiểu data cơ bản của XDR trong biểu diễn dạng chuẩn dữ liệu :

Intergers : là thực thể 32 bits . byte 0 diễn tả MSB (most significant byte) .

Variable-length opaque data : mô tả bằng chiều dài của trường ,theo sau là data .

Strings : biểu diễn bằng chiều dài ,theo sau là chuỗi ASCII bytes của một string .

Arrays :được mã bằng kích thước của trường .

Structures : được biểu diễn bằng mã hoá các thành phần của cấu trúc .

4.2 XDR Library Routines :

- Thư viện XDR cho phép người lập trình sử dụng để thực hiện các thao tác trên dữ liệu .Dưới đây là một số kiểu cấu trúc và các hàm trong thư viện XDR .Header file của thư viện này được khai báo trong <xdr.h> và khi biên dịch phải link với thư viện nsllib (dùng -lnls trong option của compiler C) .

4.2.1 Các hàm thao tác trên XDR :

Các hàm này cho phép người lập trình thao tác trên các XDR stream ,tạo buffer ,xác định vị trí trong stream .Trước khi chuyển dạng dữ liệu thì người lập trình phải tạo các XDR stream với buffer cần thiết và khi sử dụng xong thì loại bỏ các xdr stream .Các buffer tạo cho các xdr stream phải đủ và kích thước phải chia hết cho 4 bytes.

Một số hàm như sau : xdr_create, xdr_destroy, xdrmem_create, xdrrec_create, xdrstdio_create . Về chi tiết của các hàm thì xem các help của thư viện xdr .

4.2.2 Các hàm XDR đơn giản (simple xdr function) :

- Các hàm loại này cho phép chuyển đổi dữ liệu từ các dạng cơ bản như bool ,float ,double ... sang dạng chuẩn . Các hàm này cung cấp hầu như tất cả các hàm chuyển đổi từ bất kỳ dạng cơ bản của ngôn ngư C sang dạng chuẩn .

-Bao gồm một số các hàm như sau : xdr_bool , xdr_char, xdr_double, xdr_enum, xdr_float , xdr_free, xdr_hyper,xdr_int, xdr_long,xdr_longlong_t, xdr_quadruple, xdr_short, xdr_u_char, xdr_u_hyper, xdr_u_int, xdr_u_long, xdr_u_longlong_t, xdr_u_short, xdr_void .

4.4.3 Complex XDR Routine :

- Các hàm loại này cho phép chuyển đổi dữ liệu từ các dạng phức tạp được xây dựng từ các dạng cơ bản như bool ,float ,double ... sang dạng chuẩn . Các hàm này cung cấp hầu như tất cả các hàm chuyển đổi từ bất kỳ dạng phức tạp của ngôn ngư C sang dạng chuẩn .Tất nhiên là đối với những cấu trúc do user define thì user phải có các thực hiện chuyển sang dạng chuẩn .

- Bao gồm mộ số hàm sau :xdr_complex, xdr_array, xdr_bytes, xdr_opaque, xdr_pointer, xdr_reference, xdr_string, xdr_union, xdr_vector, xdr_wrapstring .

C. THIẾT KẾ VÀ HIỆN THỰC CHƯƠNG TRÌNH

1.Thiết kế và hiện thực phần hệ thống của tool :

- Phần hệ thống của tool có nhiệm vụ đảm bảo cho các vấn đề thao tác trên dữ liệu chung của các process . Ở đây chúng ta có hai phần thiết kế chính :
  - Thiết kế phần hệ thống do tool bao gồm phần hệ thống mà tool quản lý
  - Thiết kế phần hệ thống phục vụ cho các tác vụ của user task .

1.1 Phần hệ thống cho Port Server dùng quản lý thông tin :

-Trong hệ thống phân bố cũng như trong công cụ của chúng ta việc xác định vị trí của một server cũng như một tài nguyên là một phần rất quan trọng .Do vậy chúng ta trước khi truy xuất một server , hay một tài nguyên cũng như kết nối với một đối tượng nào đó thì phải xác định được vị trí của đối tượng trên hệ thống .

-Một process hay một task bất kỳ tất nhiên là không thể biết được vị trí của các task ,các process khác .Chính vì vậy khi muốn truy xuất dữ liệu dùng chung ,cũng như trao đổi dữ liệu với các task khác thì chúng phải xác định được vị trí của đối tượng thông qua tên ,port ,hay id xác định đối tượng đó .Do vậy cần phải có một server để quản lý .Các đối tượng muốn tham gia vào quá trình truyền thông trên hệ thống phải khai báo tên ,port và vị trí của chúng về server quản lý chung .Khi đó một đối tượng muốn truyền thông với một đối tượng khác hay muốn truy xuất một data chung nào đó mà không biết chúng ở đâu thì sẽ yêu cầu server quản lý chung báo cho biết . Tất cả các process trên hệ thống biết được vị trí của server quản lý chung .Tất nhiên là các server cũng phải biết được vị trí của server quản lý chung .

Trong hệ thống của chúng ta server quản lý chung quản lý năm loại như sau :

- Host ,Port của các Network Shared Memory Server (NSM Server) .
- Host ,Port của các Message Passing Daemon (MsgPassDaemon).
- Name , NSM Server (Network Shared Memory Server) nơi của các tài nguyên dùng chung (data ,var ...) .
- Name ,Host của các task tham gia vào quá trình trao đổi message thông qua message passing daemon .
- Name ,Host ,Port của các task tham gia vào quá trình trao đổi message trực tiếp .

Vì các thông tin của các server chỉ được khai báo duy nhất một lần khi khởi dộng hệ thống trong khi các thông tin còn lại được thay đổi liên tục trong quá trình thực thi của user .Do vậy việc truy xuất các thông tin của các server sẽ ít hơn so với truy xuất các thông tin còn lại và các thông tin này cũng được reset lại sau mỗi lần thực thi .Chính vì vậy chúng ta sẽ quản lý các thông tin trên theo cùng một cấu trúc nhưng không chung trong một danh sách .

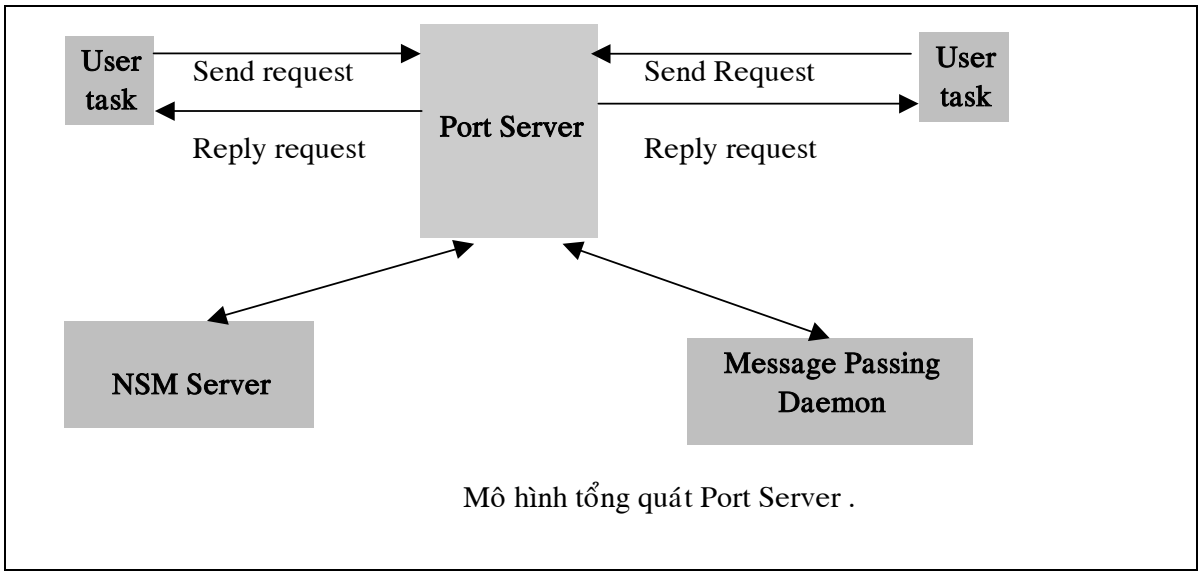
1.1.1 Mô hình tổng quát :

Trong toàn bộ thống của chúng ta có duy nhất một server quản lý chung đó là PortServer .Server này dùng quản lý thông tin của hệ thống .
Các tác vụ thao tác với Port Server bao gồm 3 tác vụ chính :

- Yêu cầu Port Server ghi thông tin .
- Yêu cầu Port Server cung cấp thông tin .
- Yêu cầu Port Server hủy thông tin .

- Server này thường xuyên được truy xuất bởi các process của hệ thống do vậy tính concurrency là cực kỳ quan trọng do vậy chúng ta thiết kế Server này là một concurrency server .

- Mô hình tổng quát của Port Server :



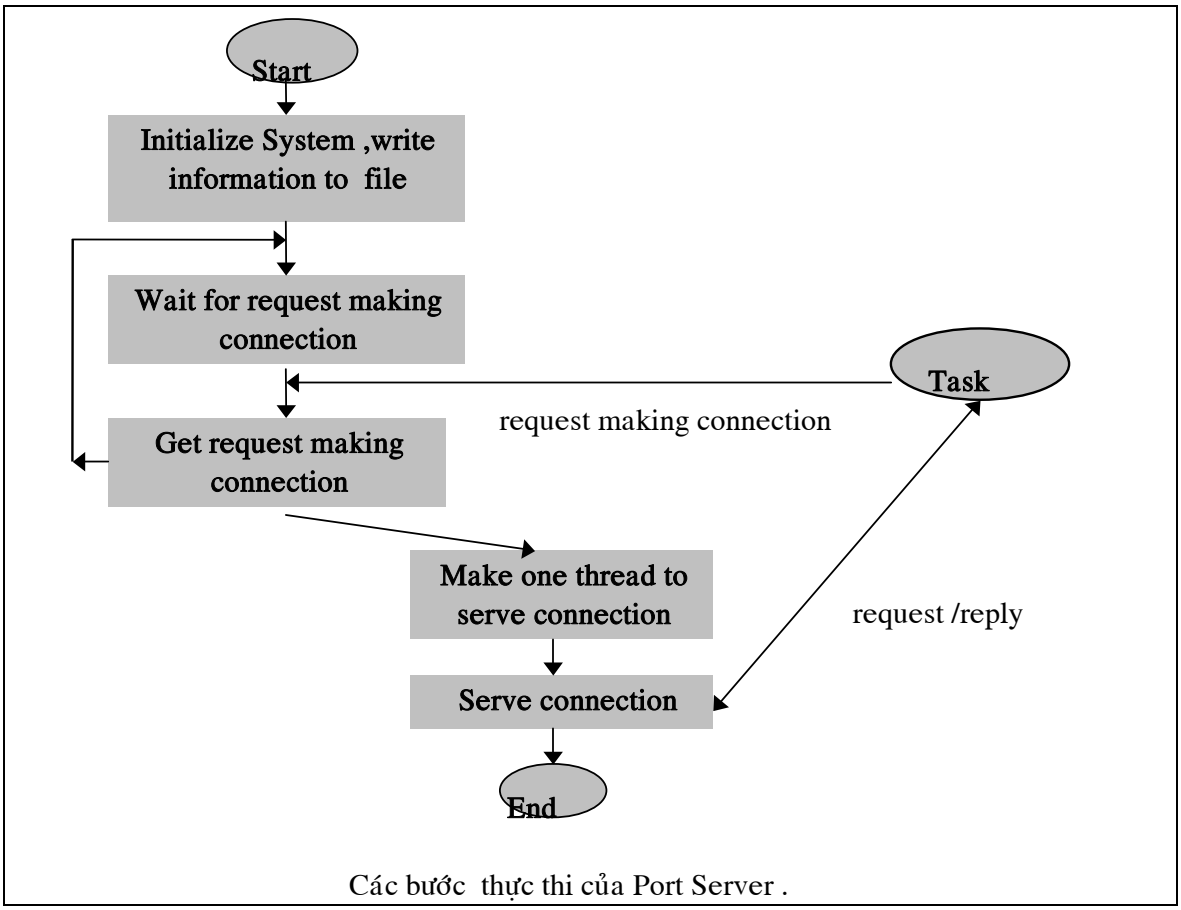
Mô hình tổng quát Port Server .

-Trong hệ thống của chúng ta Port Server là duy nhất và được tất cả các process trên hệ thống mà thực thi tại tool cần phải biết đến do vậy ta sử dụng hai biến toàn cục trong hệ thống :

```
char DPPT_HOST_PORTSERVER[20];
char DPPT_PORT_PORTSERVER[20];
```

dùng để chỉ vị trí của Port Server . Khi đưa các chương trình của user đến các máy thì chúng ta sẽ thêm vào chương trình của user các khai báo chỉ đến Port Server và khi thực thi các chương trình hệ thống thì chúng ta sẽ gởi thông tin này đến như là tham số của chương trình . Như vậy vị trí của Port Server là được biết đối với tất cả các process .

- Trong hệ thống thì server quản lý dịch vụ ,tên ,port sẽ thực hiện theo quá trình sau đây :



Các bước thực thi của Port Server .

1.1.2 Thiết kế :

1.1.2.1 Cấu trúc dữ liệu quản lý name :

```
typedef struct PortDataStruct{
        char    name_service[20] ;
                char    host_name[20];
char    host_port[20];
                PortDataStruct *next ; };
```

tên của các trường không thực sự nối lên được ý nghĩa của trường đó .Trong cấu trúc trên thì :

- name_service :dùng chỉ loại tên mà cấu quản lý .

  ⇒ Khi cấu trúc này quản lý các NSM Server ,hay MsgPassDaemon thì đơn giản là nó có cùng tên với hostname (vì hiện thực đối với một user thì chỉ có một bản có một trên một host).

  ⇒ Khi cấu trúc này quản lý các task mà tham gia trao đổi trực tiếp ,tài nguyên,biến cục bộ ,các task tham gia truyền message qua daemon thì giá trị là ID của task ,tên biến toàn cục ..

- host_name :

  ⇒ dùng để chỉ vị trí của NSM Server , MsgPassDaemon ,host của các task trao đổi trực tiếp nhau .

⇒ dùng chỉ tên tài nguyên dùng chung ,khi loại tên là tài nguyên dùng chung hay tên ID của task dùng xác định khi trao đổi message thông qua message server .

⇒ host_name của task mà trao đổi trực tiếp mesage với task khác .

- host_port :dùng chỉ port của NSM Server ,MsgPassDaemon ,user task mà trao đổi trực tiếp nhau .Cũng dùng chỉ host name đối với các global variable ,các task trao đổi thông qua MsgPassDaemon .

1.1.2.2 Cấu trúc message dùng trong truyền data :

- Cấu trúc message dùng chuyển thông tin trao đổi giữa các client và server quản lý tên như sau :

```
typedef struct PortMsgDataStruct {
        unsigned int    command ;
        unsigned int    opcode ;
        char            name_service[20] ;
        char            host_name[20];
        char            host_port[20];  };
```

- Trong đó :

◊ command :chỉ lệnh làm việc với PortServer .

◊ opcode : chỉ opcode của command .

các trường còn lại là thông tin đi kèm theo . Khi truyền message này qua hệ thống mạng thì chúng ta chuyển đổi theo dạng của network và tại nơi nhận sẽ chuyển lại . Điều này cho phép các chương trình ở trên các host khác nhau về kiến trúc vẫn thao tác được .

- Với các command như sau :

Bảng C.1: Các command trong communication với PortServer .

Command	Value	Description
DELETE_PORT	9	Yêu cầu hủy thông tin
GET_PORT	10	Yêu cầu nhận thông tin
PUT_PORT	11	Yêu cầu ghi thông tin
RESPONSE_GET_PORT	12	Báo lại yêu cầu nhận thông tin
RESPONSE_PUT_PORT	13	Báo lại yêu cầu ghi thông tin
GET_ALL_PORT	14	Yêu cầu nhận tất cả thông tin về loại nào đó .
NEXT_PORT	15	opcode báo còn thông tin tiếp theo
NO_PORT	16	opcode báo thông tin muốn nhận không có .
RESET_PORT	17	Yêu cầu reset lại thông tin .
OWN_KILL	18	Yêu cầu PortServer kết thúc.
PORT_SHARE_SERVER	19	opcode chỉ loại thông tin về NSM Server
PORT_MSG_PASS	20	opcode chỉ loại thông tin về MsgPassDaemon
PORT_RESOURCE	21	opcode chỉ loại thông tin về global variable ,global data.
PORT_NODE_DAEMON	22	opcode chỉ loại thông tin về các task truyền nhận mesage thông qua MsgPassDaemon
PORT_NODE_DIRECTLY	23	opcode chỉ loại thông tin về các user task truyền nhận trực tiếp.

1.1.2.3 Các tác vụ của Port Server :

- Để quản lý và thao tác trên dữ liệu mà Port Server quản lý chúng ta sử dụng đối tượng sau :

```
class ObjPortServer {
```

```
private :
        PortDataStruct *headPortShareServer;
        PortDataStruct *headPortMsgPassing;
        PortDataStruct *headResource;
        PortDataStruct *headNodeDaemon ;
        PortDataStruct *headNodeDirectly;
        rwlock_t       rwlock_Resource ;
        rwlock_t       rwlock_PortMsgPass ;
        rwlock_t       rwlock_PortShareServer ;
        rwlock_t       rwlock_NodeDaemon ;
        rwlock_t       rwlock_NodeDirectly ;
public :
        ObjPortServer();
        ~ObjPortServer();
        int readlock(int opcode);
        int writelock(int opcode);
        int unlock(int opcode);
        int addPort(char *name_service ,char *host_name ,char *host_port,int opcode);
        int getPort(char *name_service ,PortMsgDataStruct *portMsg,int opcode );
        int deletePort(char *name_service ,int opcode);
        int sendAllPort(int fd,int opcode);
        void resetPort(int opcode);
        int class_passiveTCP();
        void HandleSocket(int fd);
};
```

trong đó một biến cấu trúc PortDataStruct dùng để quản lý một loại thông tin và một biến readers / writer lock dùng để đồng bộ trong việc truy xuất thông tin tương ứng . Các tác vụ thao tác trên dữ liệu mà Port Server quản lý bao gồm :

- ◊ addPort : cập nhập thông tin vào trong danh sách do các task yêu cầu .
- ◊ getPort : trả lời thông tin cho các task có yêu cầu
- ◊ deletePort : loại bỏ thông tin trong danh sách .
- ◊ sendAllPort : gởi tất cả thông tin về một loại nào đó cho task có yêu cầu .
- ◊ resetPort : reset lại thông tin đã được lưu trữ .

- Các yêu cầu này được thi qua việc nhận các yêu cầu của client thông qua xử lý các message gởi đến .

-Chương trình PortServer sẽ chứa một đối tượng ObjPortServer . Phần chương trình chính của Port Server xử lý các message đến như sau :

```
ObjPortServer *pObjPortServer ;
 while (1) {
        alen =sizeof(fsin);
```

```
        slavesocket =accept(mainsocket,(struct sockaddr *)&fsin,&alen);
        if (slavesocket <0 ) {
                printf("Error in accept \n");
                continue ; }
        else {
                thr_create(NULL,0,ServerProcess,(void)&slavesocket,THR_BOUND|
THR_DETACHED , (thread_t *)NULL);
          }
        }
return 1 ;
}
```

và khi có một yêu cầu đến thì tạo một thread phục vụ cho yêu cầu của client .Thread đó phụ vụ theo đoạn chương trình sau :

```
void *ServerProcess(void *fd)
{
        int threadsocket ;
                threadsocket =*((int *)fd) ;
                pObjPortServer->HandleSocket(threadsocket);
                close(threadsocket);
                return (void *)NULL ;
}
```

- Hàm HandleSocket của object ObjPortServer sẽ giải quyết các yêu cầu của client trong vấn đề truy xuất thông tin .

1.1.3 Truy xuất PortServer :

- Như đã nói vị trí của Port Server là được tất cả các process trong hệ thống của tool cũng như các user task . Do vậy hệ thống của chúng ta xây dựng các hàm truy xuất PortServer theo các message được định nghĩa ở trên . Các hàm này được sử dụng trong việc xây dựng hệ thống và user task không cần quan tâm đến .

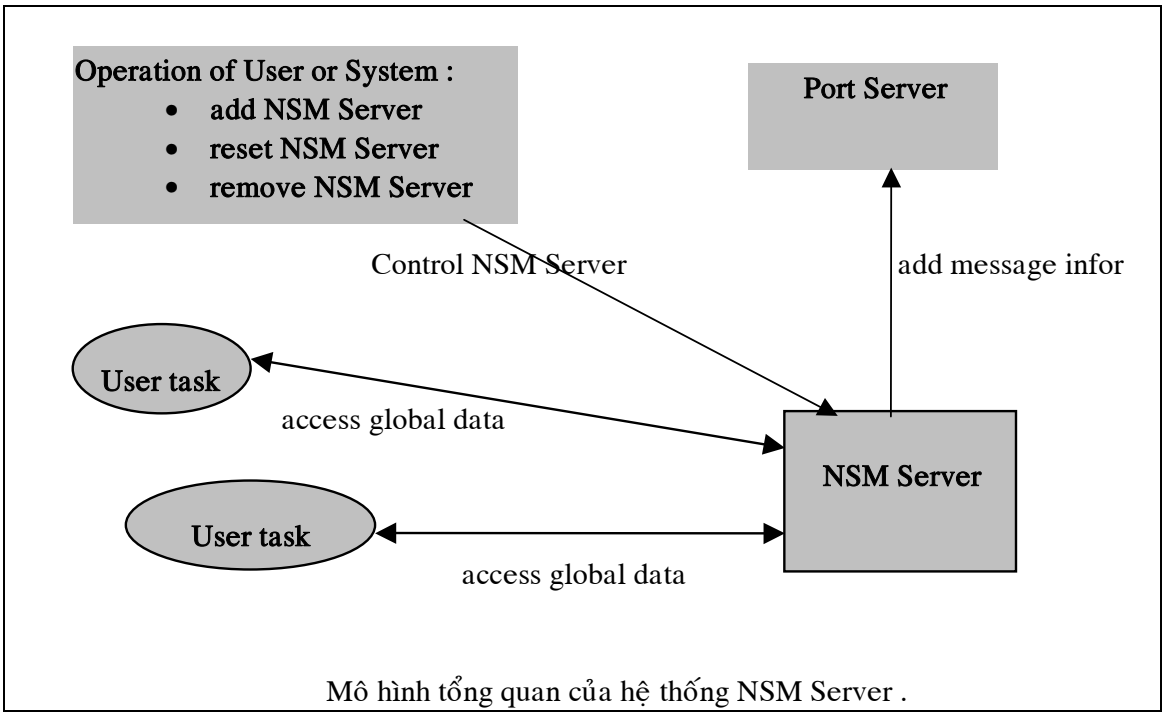
1.1.4 Các vấn đề còn lại :

- Trong việc xây dựng PortServer chúng ta còn chưa tối ưu được message truyền giữa client và server .Nếu chúng ta giảm kích thước message thì truyền sẽ đạt hiệu suất cao hơn .

1.2 Phần hệ thống cho Network Shared Memory Server (NSM Server) :

- Trong hệ thống của chúng ta sử dụng các NSM Server như là các nơi lưu trữ dữ liệu cho các user task thao tác .Để tránh tình trạng bottleneck tại một máy chúng ta dùng nhiều NSM Server trên các máy . Số lượng do người sử dụng tool quyết định và việc quản lý do PortServer đảm nhiệm cùng với một số thao tác của user .Tuy nhiên tại mỗi máy chỉ cho phép một NSM Server mà thôi

- Mô hình tổng quan của hệ thống NSM Server :



Mô hình tổng quan của hệ thống NSM Server .

- Ngoài các thao tác truy xuất dữ liệu dùng chung của user task các NSM Server còn có một số quá trình khác cho phép quản lý hệ thống như :
  - resetting : quá trình này dữ liệu chung .
  - checking system : kiểm tra xem nó có con ở trong hệ thống của tool hay không .
  - terminatin : server loại bỏ các dữ liệu mà nó giữ ,sau đó kết thúc .

1.2.1 Dữ liệu dùng chung trên Network Shared Memory Server :

-Việc xây dựng các kiểu dữ liệu cũng như các thao tác trên đó về cơ bản là như nhau .Chúng chỉ khác nhau ở mức độ chi tiết tùy theo từng loại và mục đích của chúng ta .Dữ liệu dùng chung trong hệ thống có nhiều loại dựa trên các dữ liệu cơ bản mà bất kì ngôn ngữ lập trình nào cho phép .Ở đây dữ liệu dùng chung là các kiểu dữ liệu được khai báo trong ngôn ngữ C .Trong giới hạn của bài toán chúng ta thì chúng ta chia ra một số như sau :

+Primitive Data : là những dữ liệu mà chúng ta thao tác đọc ghi ,xử lý toàn bộ không thể thao tác với phần nhỏ hơn .Loại dữ liệu này như các biến kiểu int ,bool ,float ,char ... Loại biến này chúng ta tạm dùng với danh từ là Primitive Var .

+Complex Data : dữ liệu này được tạo từ những dữ liệu cơ bản .Chúng ta có thể thao tác trên từng thành phần của nó . Tùy theo yêu cầu cũng như thời gian cho phép mà chúng ta sẽ xây dựng .Một số cấu trúc phức tạp như :

- File.
- Vector .
- Matrix ...

- Mỗi global data ,global variable được xác định bởi một tên . Trong hệ thống của chúng ta hiện tại chưa thiết kế theo kiểu phân cấp cho các global data . Chính vì thế cho nên tên xác định các global data ,global variable là duy nhất trên hệ thống của chúng ta . Vấn đề hiện thực các global data ,global variable đòi hỏi thêm nhiều kỹ thuật trong giới hạn của thiết kế này vấn đề này chưa được hiện thực .

1.2.1.1 Cấu trúc dữ liệu Primitive Data :

a. Cấu trúc của một phần tử Primitive Var :

- Để quản lý các phần tử PrimitiveVar ta sử dụng một cấu trúc dữ liệu là danh dách liên kết để quản lý . Cấu trúc của danh sách như sau :

```
typedef struct GlobalPrimitiveVarDataStruct {
        char            namePrimitiveVar[MAX_LEN_NAME_VAR] ;
        int             size ;
        unsigned int    whoaccess ;
        char            *ptrdata;
        char            setvalue ;
        rwlock_t        rwlp ;
        GlobalPrimitiveVarDataStruct *next ;
};
```

trong đó :

- *primitiveName* :tên biến .
- *size* :kích thước biến .
- *whoacess* : chỉ đến process id của user task trong hệ thống của chúng (không phải process id của OS gán cho process .) .Giá trị này dùng để xác định process nào là process truy xuất sau cùng . Sử dụng trong trường hợp khóa biến .
- *ptrdata* :con trỏ chỉ đến vùng giá trị của biến .
- *setvalue* : giá trị này bằng 1 cho biết là biến này đã được gán giá trị .
- *rwlp* : khóa multiple reader ,single writer dùng giải quyết tranh chấp trong việc đọc ghi giá trị của biến .
- *next* :chỉ phần tử kế tiếp trong danh sách .

b. Các tác vụ trên Primitive Var :

- Để cho phép các tác vụ thao tác trên Primitive Var chúng ta sử dụng một Object để quản lý dữ liệu và thực hiện thao tác trên dữ liệu . Object đó như sau :

```
class ObjGlobalPrimitiveVar {
private :
        GlobalPrimitiveVarDataStruct *headPrimitiveVar;
        rwlock_t rwlock_GlobalPrimitiveVar ;
public :
        ObjGlobalPrimitiveVar();
        ~ObjGlobalPrimitiveVar();
        void reset();
        int rdlock() {return rw_rdlock(&rwlock_GlobalPrimitiveVar); }
        int wrlock() {return rw_wrlock(&rwlock_GlobalPrimitiveVar); }
        int unlock() {return rw_unlock(&rwlock_GlobalPrimitiveVar); }
        int initGlobalPrimitiveVar(char *var_name ,int var_size,unsigned int who_access);
        int readGlobalPrimitiveVar(char *var_name,char *rectdata ,int opcode ,unsigned int
who_access);
        int writeGlobalPrimitiveVar(char *var_name ,char *inputdata ,int opcode,unsigned int
who_access);
        int deleteGlobalPrimitiveVar(char *var_name );
```

```
int deleteGlobalPrimitiveVarAll();
int lock_unlockGlobalPrimitiveVar(char *var_name ,int opcode ,unsigned int who_access);
};
```

với các tác vụ như sau :

- initGlobalPrimitiveVar (): thiết lập biến chung ,dữ liệu dùng chung .
- readGlobalPrimitiveVar() : đọc giá trị dữ liệu dùng chung ,biến chung .
- writeGlobalPrimitiveVar() : ghi giá trị dữ liệu dùng chung ,biến chung .
- lock_unlockGlobalPrimitiveVar() : khóa , mở khóa giá trị dữ liệu dùng chung ,biến chung .Có hai loại khóa là :
  - ◊ Chỉ cho đọc không cho ghi .
  - ◊ Không cho đọc /ghi .
- reset() :dùng reset lại danh sách các dữ liệu dùng chung ,biến chung .
- deleteGlobalPrimitiveVar (): loại bỏ dữ liệu dùng chung ,biến chung .
- deleteGlobalPrimitiveVarAll() : loại bỏ tất cả các dữ liệu dùng chung các biến dùng chung trong danh sách liên kết .

Cấu trúc Primitive cho phép user thao tác trên nó theo các cơ chế :

- ◊ Concurrent-Read ,Exclusive-Write (CREW)
- ◊ Concurrent-Read ,Concurrent-Write (CRCW)

- Các dữ liệu lưu trữ trên đối tượng này được truy xuất bởi nhiều thread cho nên việc truy xuất các dữ liệu của đối tượng cũng được giải quyết đồng bộ .

1.2.1.2 Cấu trúc dữ liệu Complex Data :

- Trong này ta chia ra nhiều loại .Các loại này được xây dựng từng bước .

1.2.1.2.1 Matrix :

- Ở đây ta xây dựng matrix cấp 1xN (vector) và matrix cấp NxN . Hai dữ liệu này chúng ta dùng chung một cấu trúc .

a. Cấu trúc của một phần tử matrix :

- Để quản lý matrix ta cũng dùng danh sách liên kết để quản lý . Cấu trúc như sau :

```
typedef struct GlobalMatrixDataStruct {
        char            matrixName[MAX_NAME_MATRIX] ;
        int             elementSize ;
        int             rowSize ;
        int             colSize ;
        unsigned int    whoaccess ;
        char            *ptrMatrixData;
        rwlock_t        rwlp;
        GlobalMatrixDataStruct *nextMatrix ;
};
```

trong đó :

- *matrixName* :tên ma trận,vector chiều dài tối đa là MAX_NAME_MATRIX bằng 20 bytes.
- *elementSize* :kích thước một phần tử ma trận .
- *rowSize* :số hàng ma trận .
- *colSize* :số cột ma trận ,nếu vector thì giá trị của trường này là 0 .

- *whoacces* : chỉ đến process id của user task trong hệ thống của chúng (không phải process id của OS gán cho process .) .Giá trị này dùng để xác định process nào là process truy xuất sau cùng .
- *ptrMatrixData* : chỉ đến vùng dữ liệu của matrix .
- *rwlp* :giống như loại biến trên .
- *nextMatrix* :chỉ đến phần tử kế tiếp trong list .

b. Các tác vụ trên kiểu Matrix :

- Để cho phép các tác vụ thao tác trên Matrix chúng ta cũng sử dụng một Object để quản lý các dữ liệu và thao tác trên dữ liệu đó . Object đó như sau :

```
class ObjGlobalMatrix {
private :
GlobalMatrixDataStruct *headMatrix ;
                rwlock_t rwlock_GlobalMatrix ;
 public :
                ObjGlobalMatrix();
                ~ObjGlobalMatrix();
                void reset();
                int rdlock() { return rw_rdlock(&rwlock_GlobalMatrix);}
                int wrlock() { return rw_wrlock(&rwlock_GlobalMatrix);}
                int unlock() { return rw_unlock(&rwlock_GlobalMatrix);}
                int make_NewMatrix(char *matrix_name ,int row_number ,int col_number ,int
                elementSize,unsigned int who_access ,GlobalMatrixDataStruct *newMatrix);
                int initGlobalMatrix(char *matrix_name ,int row_number,int col_number ,int
                elementSize,unsigned int who_access);
                int removeGlobalMatrix(char *matrix_name);
                int removeGlobalMatrixAll();
                int writeSubMatrix2(char *matrix_name , char *inputdata,int sRow,int sCol,int
                eRow,int eCol,unsigned int who_access);
                int readSubMatrix2(char *matrix_name , char *rectdata,int sRow,int sCol,int
                eRow,int eCol,unsigned int who_access) ;
                int lock_unlockGlobalMatrix(char *matrix_name ,int opcode,unsigned int
                who_access);
} ;
```

với các tác vụ như sau :

- initGlobalMatrix (): thiết lập một matrix dùng chung .
- readSubMatrix2() : đọc giá trị submatrix của một matrix .
- writeSubMatrix2() : ghi giá trị submatrix của một matrix .
- lock_unlockGlobalMatrix() : khóa , mở khóa giá trị dữ liệu dùng chung ,biến chung .Có hai loại khóa là :
  - ◊ Chỉ cho đọc không cho ghi .
  - ◊ Không cho đọc /ghi .
- reset() :dùng reset lại danh sách các matrix ,vector .
- removeGlobalMatrix (): loại bỏ một matrix ,vector ra khỏi danh sách .
- removeGlobalMatrixAll() : loại bỏ tất cả các matrix ,vector trong danh sách liên kết

- Đối với kiểu dữ liệu matrix trong quá trình hiện thực tại thời điểm này cần lưu ý sau đây :

◊ Vấn đề đọc ghi một matrix hiện tại hiện thực theo cơ chế Concurrent-Read, Concurrent -Write (CRCW).

◊ Do đó việc đọc ghi cần chú ý đến tính đúng đắn của dữ liệu .

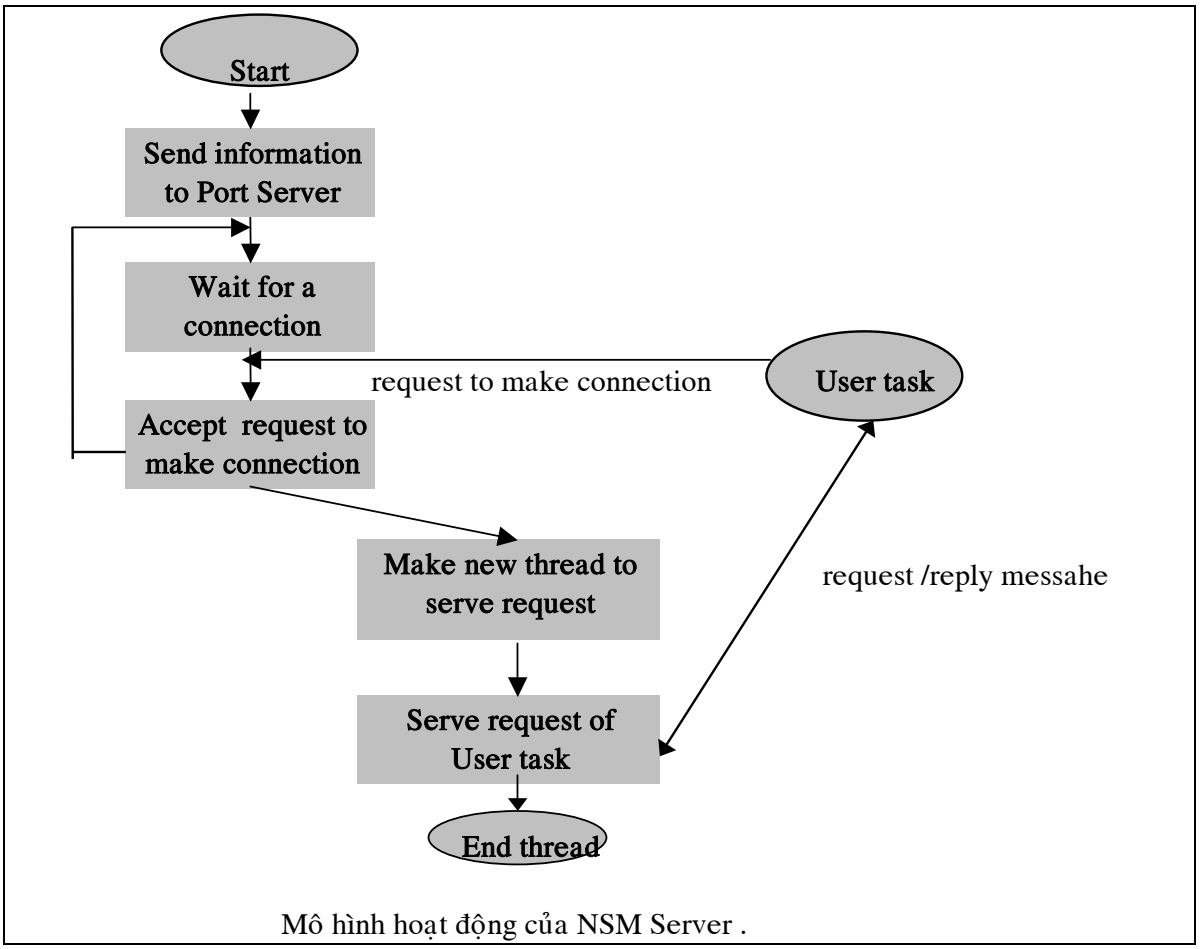
1.2.1.2.2 File :

- Về file trên server chúng ta không xây dựng cấu trúc dữ liệu để quản lý file .Khi dữ liệu dùng chung là file thì chúng ta thao tác trực tiếp trên file . Các thao tác trên file như sau :

- put ShareFile() : user task gởi một file về NSM Server . NSM Server lưu sẽ file lại
- get ShareFile() : user task yêu cầu nhận một file từ NSM Server . NSM Server sẽ gởi file cho user task .

1.2.2 Thiết kế Network Shared Memory Server (NSM Server):

1.2.2.1 Chương trình NSM Server:

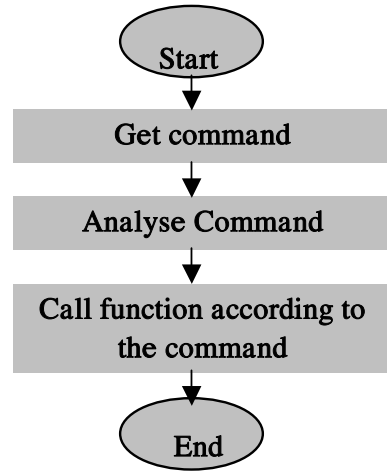


Mô hình hoạt động của NSM Server .

- Chương trình NSM Server là chương trình dạng daemon .Chương trình này có nhiệm vụ quản lý các tài nguyên,dữ liệu chung mà user thiết lập trên hệ thống . Server này cho phép các tác vụ cài đặt ,đọc ,ghi và hủy dữ liệu dùng chung .Vì server này có nhiều process cùng truy xuất cho nên đòi hỏi server phải đáp ứng nhanh các tác vụ .Vì những lý do trên nên server này được thiết kế là một concurrency server .

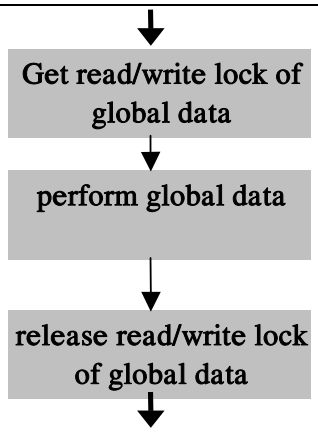
-Mỗi khi có một cầu nối từ process client yêu cầu server thì server sẽ tạo ra một thread mới .Thread này có nhiệm vụ thực hiện hàm ServerProcess phục vụ những yêu cầu của process client

cho đến khi nào cầu nối này kết thúc .Dưới đây là sơ đồ các bước thực hiện của hàm ServerProcess :



Các bước của hàm ServerProcess (xử lý command trên NSM Server)

- Tùy theo các loại message mà client yêu cầu server thì server sẽ thực hiện các lệnh do client gởi tới .
- Dữ liệu dùng chung được lưu trữ trên từng server được truy xuất bởi nhiều thread phục vụ các yêu cầu của client do vậy để đảm bảo được tính đồng bộ của dữ liệu thì mỗi loại tài nguyên ,dữ liệu chung phải có một biến dùng để đồng bộ . Quá trình truy xuất dữ liệu dùng chung của các thread như sau :



Quá trình truy xuất data chung của các thread trên NSM Server .

Về chi tiết của việc truy xuất tài nguyên chung chúng ta sẽ nối rõ hơn trong phần sau .
-Riêng các phần dùng để cho hệ thống quản lý thì bao gồm :

- resetting : với chức năng này thì khi NSM Server nhận được yêu cầu tự reset thì nó sẽ reset lại toàn bộ dữ liệu mà nó lưu trữ . Một đặc tính trong mô hình DSM đó là dữ liệu dùng chung có thể được lưu lại cho như mục đích sau này . Do vậy quá trình yêu cầu

NSM Server reset chỉ xảy ra khi user tạo,hay mở một ứng dụng mới mà thôi .Điều này cho phép dữ liệu được tính toán bởi một phần của ứng dụng có thể được sử dụng lại .

- checking system : chức năng này dùng để tự NSM Server kiểm tra xem chúng có còn nằm trong hệ thống quản lý của tool hay không . Một khi chúng không còn nằm trong hệ thống quản lý của tool thì chúng sẽ tự kết thúc .
- terminating :với chức năng này thì NSM Server tự loại bỏ các tài nguyên mà nó quản lý ,sau đó sẽ kết thúc quá trình thực thi khi nhận được yêu cầu .

-Để tạo linh động thì user có thể thực thi nhiều NSM Server như thế trên hệ thống tuy nhiên chúng ta sẽ thiết kế sao cho tại mỗi máy chỉ có một NSM Server thực thi mà thôi .Kỹ thuật thiết kế để duy nhất một bản đang thực thi trên một máy sẽ trình bày ở phần sau :

- Đối tượng Hiện thực của chương trình NSM Server như sau :

```
class ObjNSMServer {
  private :
    ObjGlobalPrimitiveVar *pShareGlobalPrimitiveVar ;
    ObjGlobalMatrix *pShareGlobalMatrix ;
    mutex_t mutex_tty ;
    void HandleMessageInitPrimitiveVar(HeadCommand *requestCommand ,int fd);
    void HandleMessageReadPrimitiveVar(HeadCommand *requestCommand ,int fd);
    void HandleMessageWritePrimitiveVar(HeadCommand *requestCommand ,int fd);
    void HandleMessageRemovePrimitiveVar(HeadCommand *requestCommand ,int fd);
    void HandleMessageRemovePrimitiveVarAll(HeadCommand *requestCommand ,int fd);
    void HandleMessageServerAlive(HeadCommand *requestCommand ,int fd);
    void HandleMessageLock_unlockPrimitiveVar(HeadCommand *requestCommand ,int fd);
    void HandleMessageInitMatrix(HeadCommand *requestCommand,int fd);
    void HandleMessageReadSubMatrix(HeadCommand *requestCommand ,int fd);
    void HandleMessageWriteSubMatrix(HeadCommand *requestCommand,int fd);
    void HandleMessageRemoveMatrix(HeadCommand *requestCommand,int fd);
    void HandleMessageRemoveMatrixAll(HeadCommand *requestCommand,int fd);
    void HandleMessagePutFile(HeadCommand *requestCommand ,int fd);
    void HandleMessageGetFile(HeadCommand *requestCommand,int fd);
    void HandleMessageDefault(HeadCommand *requestCommand,int fd);
    void HandleMessageReset(HeadCommand *requestCommand ,int fd);
  public :
    ObjNSMServer();
    ~ObjNSMServer();
    void HandleMessageSocket(int fd);
};
```

- Mỗi ObjServer quản lý hai đối tượng lưu trữ global data ,global variable là:

ObjGlobalPrimitiveVar

ObjGlobalMatrix

hàm HandleMessageSocket() nhận một socket và xử lý các yêu cầu đến với socket đó .

-Chương trình NSM Server chứa một đối tượng chung kiểu ObjNSMServer . NSM Server chờ đợi yêu cầu như sau :

```
ObjNSMServer *pNSMObjServer ; // global object
```

chờ yêu cầu :

```
while (DOWORK) {
alen =sizeof(fsin);
        slavesocket =accept(mainsocket,(struct sockaddr *)&fsin,&alen);
        if (slavesocket <0 )
                printf("Error in accept \n");
        else {
        thr_create(NULL,0,ServerProcess,(void *) &slavesocket ,THR_BOUND |
THR_DETACHED , ( thread_t *) NULL);
        }
```

Khi có một yêu cầu tới thì NSM Server tạo ra một thread thực thi hàm ServerProcess như sau :

```
void *ServerProcess(void *fd)
{       int threadsocket ;
                threadsocket =*((int *)fd) ;
                pNSMObjServer->HandleMessageSocket(threadsocket);
                close(threadsocket);
                return (void *)NULL ;
}
```

toàn bộ vấn đề xử lý được thực thi trong hàm HandleMessageSocket của đối tượng ObjNSMServer.

1.2.2.2 Vấn đề truyền thông giữa client và server :

- Để phục vụ cho việc truyền các yêu cầu cũng như data giữa client và server ta dùng socket interface của TCP/ IP protocol .Khi một server thực thi thì server sẽ báo cho hệ thống của chúng ta vị trí máy và port mà server đang lắng nghe bằng cách gởi thông tin về Port Server .Các thông tin này sẽ được dùng vào các mục đích sau :

    - Cho client biết vị trí server .
    - Dùng quản lý hoạt động của server như reset server ,kill server ...
    -

- Các bước hoạt động của server như sau :

a. Các cấu trúc message cho quá trình truyền thông :

-Các loại message dùng để truyền thông giữa client và server như sau :

i) Command Message :

```
        typedef struct HeadCommand {
                        unsigned int command ;
                        unsigned int opcode ;
                }
```

trong đó :

- command : dùng chỉ loại command .
- opcode :dùng chỉ mã kèm theo của command .

-Trong chương trình của chúng ta vì nhiều lý do chỉ sử dụng một số tài nguyên dùng chung do đó chúng ta chỉ có một số command .Mỗi command sẽ có một ID và một giá trị tương ứng .
- Dưới đây là bảng các command trong trao đổi dữ liệu dùng chung trên NSM Server :
Bảng C.2 : Các command trong truyền nhận global data của NSM Server

Command	Value	Description
NEXT_DATA	90	chỉ còn data gởi tiếp
RESPONSE_NEXT_DATA	91	báo lại yêu cầu trên
END_NEXT_DATA	92	báo không còn data tiếp theo
READ_PRIMEVAR	100	yêu cầu đọc biến PrimitiveVar
RESPONSE_READ_PRIMEVAR	101	báo lại command đọc biến
WRITE_PRIMEVAR	102	yêu cầu ghi biến PrimitiveVar
RESPONSE_WRITE_PRIMEVAR	103	báo lại ghi biến PrimitiveVar
INIT_PRIMEVAR	104	yêu cầu thiết lập biến PrimitiveVar
RESPONSE_INIT_PRIMEVAR	105	báo lại yêu cầu trên
REMOVE_PRIMEVAR	106	yêu cầu loại biến PrimitiveVar
RESPONSE_REMOVE_PRIMEVAR	107	báo lại yêu cầu trên
REMOVE_PRIMEVAR_ALL	108	yêu cầu loại tất cả các biến PrimitiveVar
RESPONSE_REMOVE_PRIMEVAR_ALL	109	báo lại yêu cầu yêu trên
LOCK_UNLOCK_PRIMEVAR	110	yêu cầu lock /unlock biến PrimitiveVar
RESPONSE_LOCK_UNLOCK_PRIMEVAR	111	báo lại yêu cầu trên
SERVER_ALIVE	112	kiểm tra server có còn hay không
RESPONSE_SERVER_ALIVE	113	báo lại yêu cầu trên
INIT_MATRIX	120	yêu cầu thiết lập một matrix
RESPONSE_INIT_MATRIX	121	báo lại yêu cầu trên
READ_SUB_MATRIX	128	yêu cầu đọc một submatrix của một matrix
RESPONSE_READ_SUB_MATRIX	129	báo lại yêu cầu trên
WRITE_SUB_MATRIX	130	yêu cầu ghi vào một submatrix của một matrix
RESPONSE_WRITE_SUB_MATRIX	131	báo lại yêu cầu trên
REMOVE_MATRIX	132	yêu cầu loại bỏ một matrix
RESPONSE_REMOVE_MATRIX	133	báo lại yêu cầu trên
REMOVE_MATRIX_ALL	134	yêu cầu loại bỏ tất cả các matrix
RESPONSE_REMOVE_MATRIX_ALL	135	báo lại yêu cầu trên
LOCK_UNLOCK_MATRIX	136	yêu cầu lock /unlock matrix
RESPONSE_LOCK_UNLOCK_MATRIX	137	báo lại yêu cầu trên
PUT_FILE	138	yêu cầu put file lên NSM Server
RESPONSE_PUT_FILE	139	báo lại yêu cầu trên
GET_FILE	140	yêu cầu lấy về một file trên NSM Server
RESPONSE_GET_FILE	141	báo lại yêu cầu trên
OWN_SERVER_KILL	146	yêu cầu server kết thúc
OWN_SERVER_RESET	147	yêu cầu server reset .

ii) Infor Message :

- Tùy theo từng loại data dùng chung mà chúng ta sẽ sử dụng các loại infor message khác nhau . Các message này dùng truyền các thông tin đi kèm với command message . Chúng ta sử dụng các loại infor message dưới đây :

- Infor Message về các Global PrimitiveVar .

- Infor Message về các Global Matrix .
- Infor Message về File .

Cấu trúc của các infor message như sau :

```
#define MAXLEN_NAME 20
typedef struct InforPrimitiveVar {
        char mesg_name[MAXLEN_NAME] ;
        int  mesg_size ;
        int  mesg_opcode ;
};

typedef struct InforMatrix {
        char matrixName[MAXLEN_NAME];
        int  sRow ;
        int  sCol ;
        int  eRow ;
        int  eCol ;
        int  elementSize;
};

typedef struct InforFile {
        char fileName[MAXLEN_NAME];
        int fileSize ;
};
```

- Để bảo đảm cho việc truyền thông giữa các máy khác nhau thì các loại message trên trước khi truyền đi đều được chuyển sang dạng format XDR và sau đó tại nơi nhận sẽ chuyển ngược trở lại .

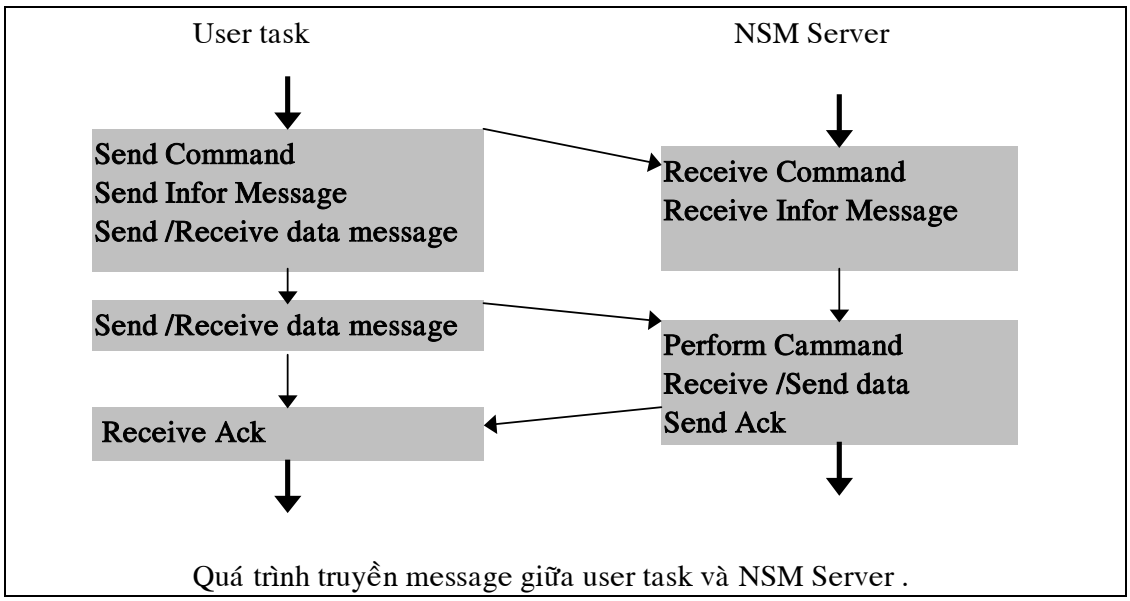
iii) Data Message :

- Data message là một dạng data nào đó dưới dạng chuỗi data này sẽ được truyền qua lại giữa client và server . Data này được nhận dạng dựa vào các message trước đó .
-Để nhận dạng dúng kiểu data và kích thước của data thì giữa sender và receive đã thỏa thuận nhau bằng các command message & infor messgase .Do đó trước khi nhận hay truyền một message data thì bên còn lại đã biết kích thước của dữ liệu nhận .
- Khác với các loại message trên ,đối với message data thì data là do user đưa vào đó vậy tool không chuyển sang dạng chuẩn mà để cho user làm điều đó .

1.2.2.3 User task truy xuất data chung :

- Một user task thao tác với data chung trên NSM Server bao gồm các thao tác chính sau đây:
  - khởi động hệ thống cần thiết để tham gia vào quá trình truy xuất NSM Server
  - initialize global data : lúc này tool phải có nhiệm vụ xác định server nào sẽ cài đặt tài nguyên chung cho user task trên hệ thống của chúng ta .
  - các thao tác đọc ,ghi ,xoá ... thì lúc này tool phải có nhiệm vụ xác định được vị trí của tài nguyên để thao tác .

- Để user task có thể tham gia vào quá trình truy xuất trên NSM Server thì hệ thống phải có ít nhất một NSM Server đang thực thi .
- Quá trình truyền nhận message giữa user task và NSM Server như sau :



Quá trình truyền message giữa user task và NSM Server .

- Bất kì một tác vụ nào của user task mà truy xuất NSM Server điều thực hiện các bước trên .Hệ thống của chúng ta sẽ đảm bảo điều đó .

- Để quản lý các global variable ,các NSM Server cho user task chúng ta sử dụng một object . Object này có nhiệm vụ lưu trữ các vị trí của NSM Server và các global data .global variable . Khi cần thiết đối tượng này sẽ liên lạc với PortServer để nhận thông tin .

- Đối tượng đó như sau :

```
typedef struct ObjServerStruct {
                char host_name[20] ;
                char host_port[20] ;
} ;

class ObjServer {
        private :
                ObjResource   *objResource ;
                ObjResource   *objRShareServer ;
                ObjServerStruct objServerStruct[MAX_SERVER];
                int serverNumber ;

        public :

                ObjServer();
                ~ObjServer();
```

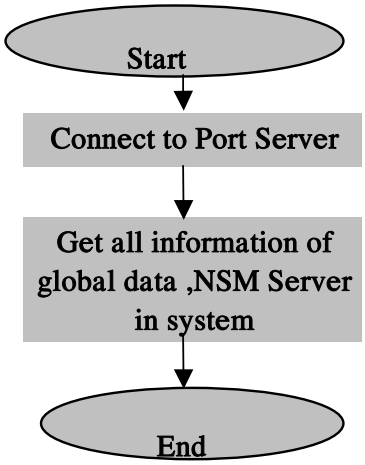
```
        int getSocket(char *resourceName) ;
        int connect(int servernumber);
        int GetInforServer(char *host_name);
        int getserver(int servernumber ,char *host_name);
        int getServerNumber() { return serverNumber ; }
};
```

- Object này có nhiệm vụ là nhận tên một global data ,globa variable và trả về một cầu nối socket chỉ đến NSM Server nơi chứa data đó . Khi đó sẽ cho phép truy xuất đến NSM Server thông qua cầu nối đó .

- Trong Object trên có chứa một Object khác là ObjResource . ObjResource là một đối tượng lưu trữ các thông tin về một loại nào đó . ObjResource có thể liên lạc với NSM Server để thêm bớt thông tin . ObjServer chứa hai ObjResource : một lưu trữ cho thông tin về các NSM Server và một lưu trữ thông tin cho global variable ,global data .

a) Khởi động hệ thống cho user task truy xuất NSM Server :

- Quá trình khởi động hệ thống của user task có mục đích là báo cho các chương trình hệ thống biết user task tham gia vào truy xuất trên các NSM Server . Từ đó hệ thống sẽ tạo các biến hệ thống ,gởi các thông tin cần thiết .Khi user task truy xuất thì có sẵn một số thông tin ban đầu . Thực hiện điều này cho phép user linh động khi cần truy xuất và làm giảm đi các tài nguyên của hệ thống dành cho user task .



Quá trình khởi động của một user task mà trao đổi data qua NSM Server

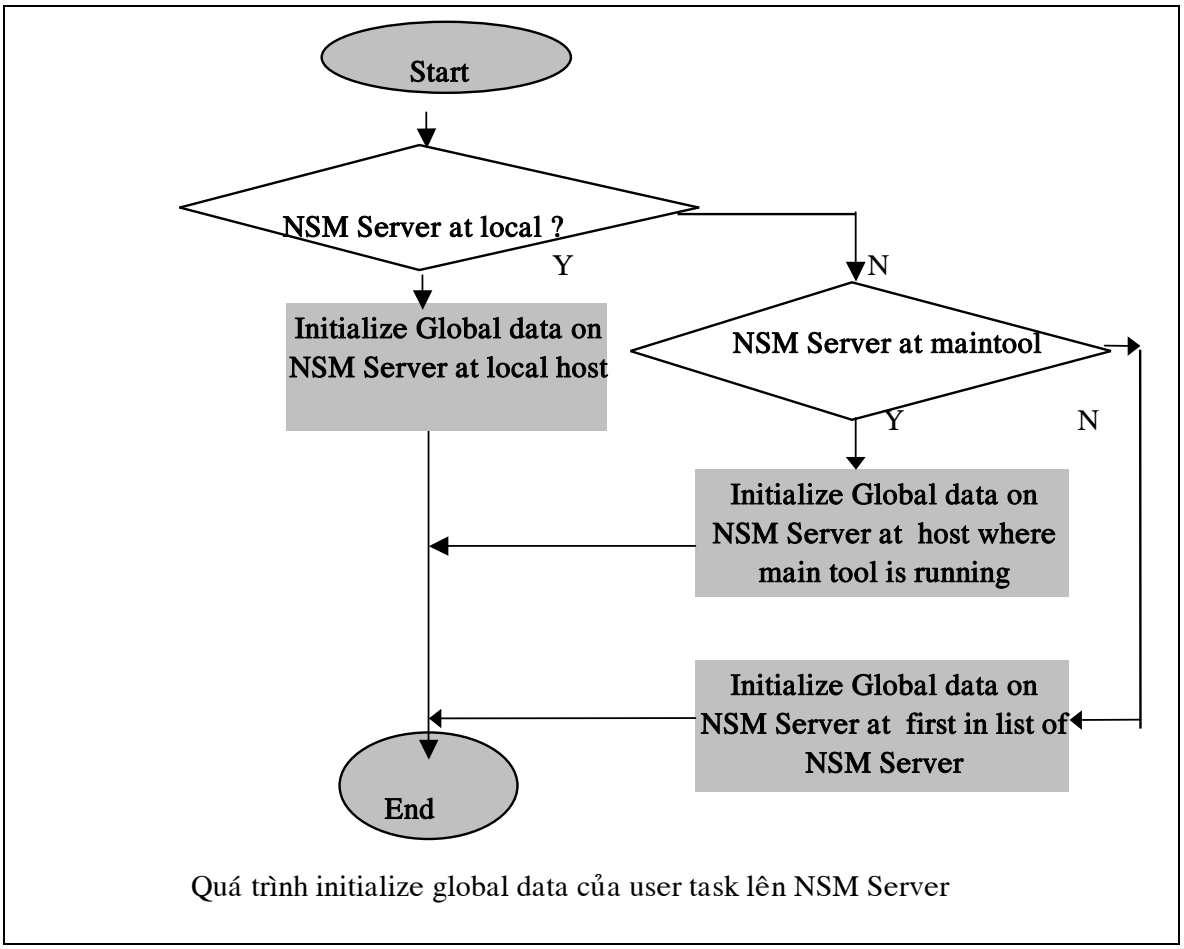
- Khi User task khai báo initialize thì chúng ta tạo ra một biến kiểu ObjServer

```
ObjServer *objServer =new ObjServer() ;
```

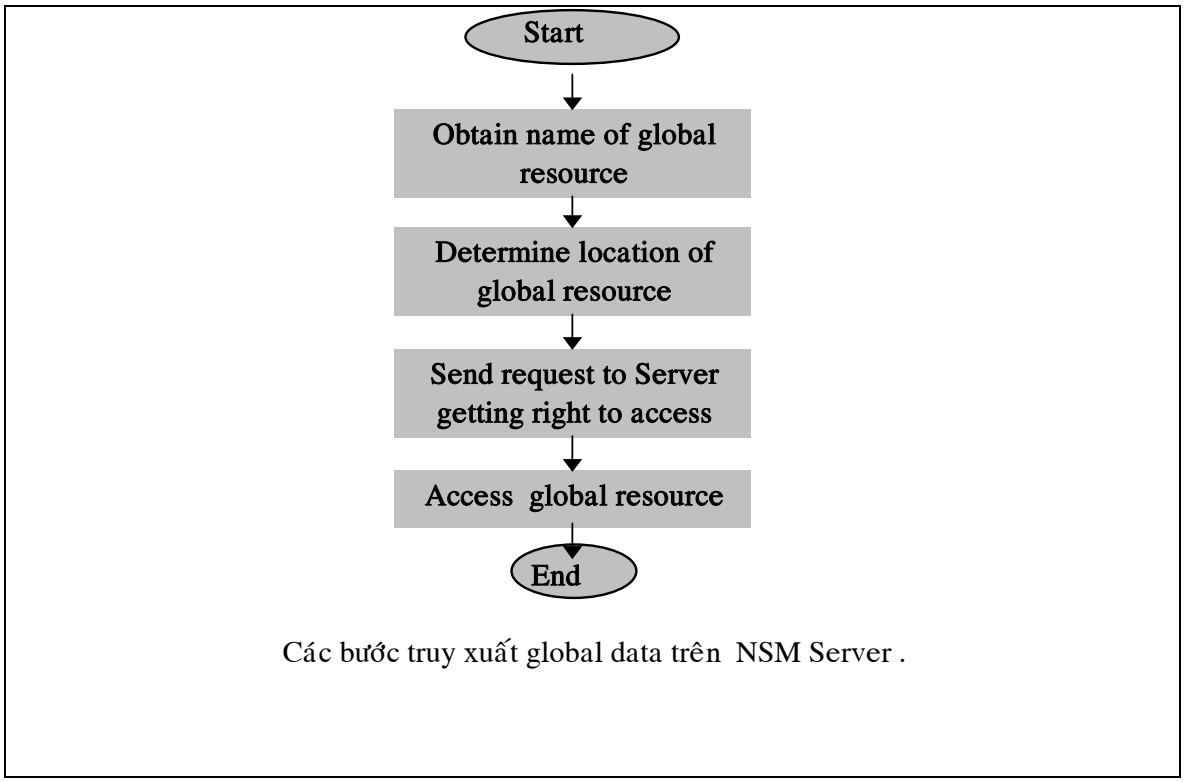
Khi khởi tạo đối tượng này sẽ liên lác với PortServer để nhận thông tin và user task sẽ có đủ thông tin về các NSM Server hiện tại và một số thông tin về các global variable ,global data nếu có .

b) Quá trình thiết lập tài nguyên chung :

- Quá trình thiết lập biến chung ,dữ liệu dùng chung trên NSM Server theo giải thuật dưới đây là vì hệ thống của chúng ta có một hoặc nhiều NSM Server . Chúng ta cũng không thể nào biết được thiết lập trên NSM Server nào là tốt nhất . Vì điều này liên quan đến nhiều vấn đề mà chúng ta không nắm hết cho nên chúng ta sẽ theo nguyên tắc đơn giản cho rằng NSM Server tại vị trí mà user task là nơi tốt nhất . Và sau đó là các vị trí khác .



Quá trình initialize global data của user task lên NSM Server



Các bước truy xuất global data trên NSM Server .

c) Quá trình thao tác trên tài nguyên chung :

-Khi một user task muốn truy xuất biến chung ,data chung trên NSM Server thì user task sẽ sử dụng các hàm thư viện và thông qua tên của tài nguyên sẽ truy xuất . Các thư viện hệ thống sẽ bảo đảm cho quá trình truy xuất của user task .

1.2.3 Các vấn đề còn lại :

- Việc hiện thực NSM Server hiện tại đang còn một số vấn đề như sau :
  - ◊ Vấn đề dữ liệu truyền của user lớn . Điều này bắt buộc chúng ta phải cắt dữ liệu của user thành những phần nhỏ hơn để truyền .
  - ◊ Trong khi hiện thực phần này chúng ta không tự động chuyển đổi dữ liệu của user mà cho user tự đổi điều này có ưu điểm là tránh lãnh phí . Nhưng ngược lại thì bắt buộc user phải thao tác để xử lý .

1.3 Phần hệ thống cho Message Passing :

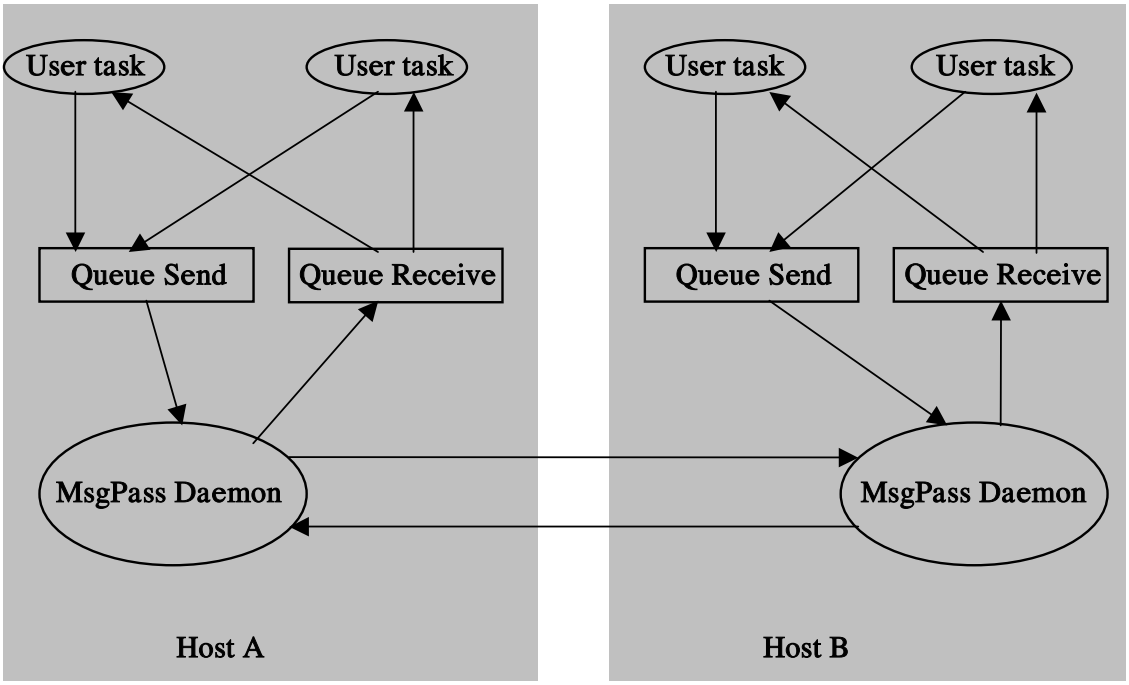
- Có hai cách truyền message là :
  - Các task truyền message thông qua các daemon .
  - Các task truyền message trực tiếp nhau .

trong công cụ của chúng ta thì thiết kế cho phép các user task sử dụng cả hai phương pháp trên để truyền message . Mỗi phương pháp đòi hỏi hệ thống phải quản lý cũng như có những thao tác khác nhau .

1.3.1 Message passing thông qua Message Passing Daemon (MsgPassDaemon):

1.3.1.1 Mô hình tổng quan :

- Trong chương trình của chúng ta để xây dựng phần truyền thông điệp thì như chúng ta có đề cập về mô hình message passing thì ta có mô hình sau :



Mô hình tổng quan truyền Message Passing

- Từng máy có trên hệ thống thực thi chương trình message passing daemon .Chương trình daemon này quản lý một tập database về các task trong hệ thống cũng như các message passing daemon khác trong hệ thống .Server này đồng thời cũng tạo ra hai message queue :

- message queue for sending .
- message queue for receiving.

một queue cho các message được gởi đi và một queue cho các message nhận được . Các queue này là các message queue mà hệ thống cung cấp trong IPC .Khi một process muốn gởi message cho một process thì process đó sẽ gởi message vào message queue send .Server sẽ nhận các message từ message queue send và xác định vị trí đích của message dựa vào database của nó

.Khi đó server sẽ thực hiện kết nối với message passing server ở nơi đến và gởi message đó đi . Tại nơi nhận sẽ nhận message và đưa vào trong message queue receive . Khi một process muốn nhận message thì nó sẽ đọc message queue receive để tìm ra message dành cho nó .

- Như vậy so với mô hình NSM Server thì tại mỗi máy mà có user task truyền nhận message qua MsgPassDaemon phải có chương trình MsgPassDaemon thực thi .

1.3.1.2 Các queue và quản lý truy xuất queue :

- Các sending queue và receiving queue là các message queues của hệ thống do vậy tất cả các user task và MsgPassDaemon trên một máy điều có thể truy xuất được . Mỗi queue có một key xác định .Hệ thống của chúng ta sẽ tạo key duy nhất cho mỗi queue đảm bảo rằng không có việc trùng lắp các key . Key được tạo ra chung cho cả MsgPassDaemon và các hàm hệ thống của thư viện cho user task .

- Các message queues được tạo ra khi chương trình MSgPassDaemon thực thi . Đối với các user task thì chúng không tạo các message queues mà chúng chỉ nhận lấy rồi sử dụng thôi .

- Mỗi message chứa trong queue bao gồm data của user và thêm vào đó là các thông tin thêm vào cho biết nơi gởi và nơi nhận .

-Cấu trúc data của user message :

```
#define HEADERMSG 8
#define MAXMESGDATA 1024
        typedef struct UserInfor {
                        unsigned int mesg_id ;
                        unsigned int mesg_len ;
                        char mesg_buf[MAXMESGDATA - HEADERMSG];
                } ;
-Cấu trúc message chứa trong queue :
        typedef struct MesgPassInfor {
                long mtype ;                    // type of message .
                char src_nodename[20];          // source nodename (sender )
                char des_nodename[20];          // destination nodename (receiver)
        UserInfor       data ;
        } ;
```

i) Object quản lý message queues :

- Để quản lý các tác vụ đọc ghi của chương trình MsgPassDaemon cũng như của user task đối với các message queues ta sử dụng Object sau để quản lý các tác vụ truy xuất message queues

```
class  QueuMessage {
        private :
                int queuLength ;
                int queuMax ;
                int queuId ;
                int queuOpcode;
                key_t queuKey ;
                ObjRtt *objRtt ;
        public :
                QueuMessage(int key);
                QueuMessage(int key ,int opcode);
                ~QueuMessage() ;
                int reset();
                int pushMsg(MesgPassInfor *infor);
                int pushMsg(MesgPassInfor *infor ,short timeout);
                int popMsg(MesgPassInfor *infor);
                int popMsg(MesgPassInfor *infor, short timeout);
```

};

với hai tác vụ chính :

- pop()
- push()

- Để user task có thể truy xuất đúng message của mình thì mỗi message sẽ có một key duy nhất . Key này được tạo ra theo giải thuật sau :
  - Khi gởi : key được tạo ra từ tên của task gởi và tên của task nhận .Từ chuỗi src_nodename và des_nodename ta tạo ra key cho message đó . Khi gởi chuỗi tạo key là :
    des_nodenamesrc_nodename .
  - Khi nhận :thì sẽ nhận các message có key là key được tạo từ tên của task nhận và tên của task gởi . Quá trình này ngược lại với quá trình trên . Chuỗi tạo key để nhận là :
    src_nodenamedes_noname

- Khi chương trình MsgPassDaemon khai báo đối tương queue thì MsgPassDaemon sử dụng lệnh :

```
QueuMessage *queuSend =new QueuMessage(0,1) ;
QueuMessage *queuSend =new QueuMessage(1,1) ;
```

còn đối với user task thì hệ thống sẽ khai báo :

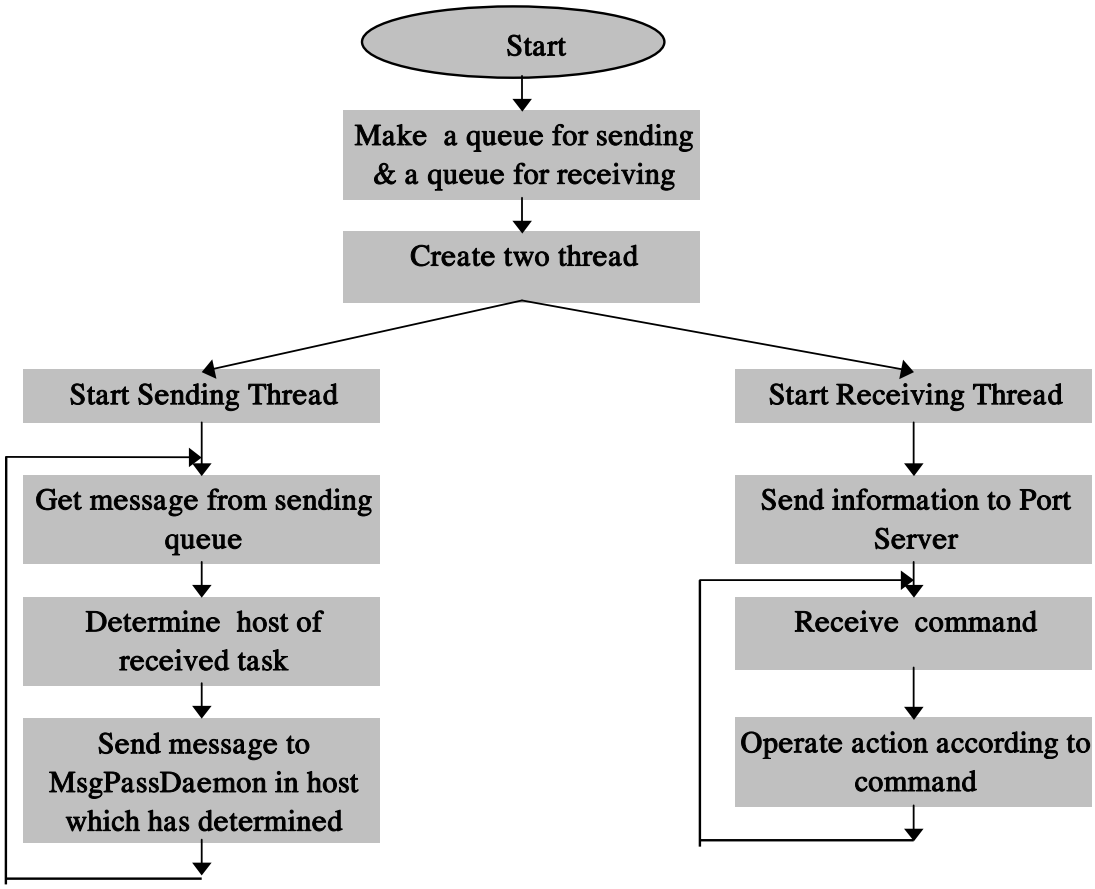
```
QueuMessage *queuSend =new QueuMessage(0) ;
QueuMessage *queuSend =new QueuMessage(1) ;
```

-Các tác vụ popMsg() và pushMsg() của đối tượng trên thực chất là gọi các tác vụ msgsnd() và msgrcv() của hệ thống . Chương trình sẽ tạo key duy nhất cho mỗi queue và không trùng lặp cho các user khác nhau .Điều đó đảm bảo cho việc multiuser của tool .

1.3.1.3 Thiết kế Message Passing Daemon (MsgPassDaemon) :

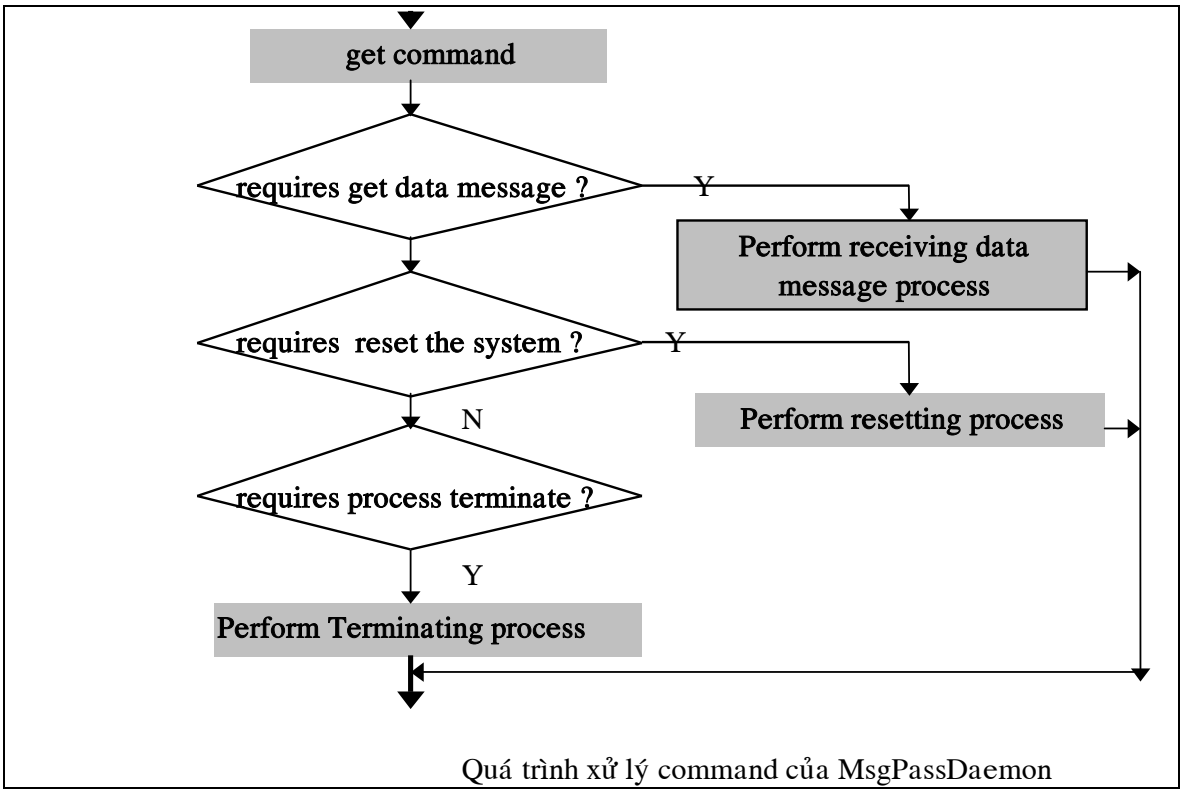
-Mỗi Message Passing Daemon chứa dựng hai server nhỏ hơn là send server và receive server .Giống như khi thiết kế NSMS .Message Passing Daemon cũng là một concurrency program với hai thread : một thực hiện chức năng sender và một thực hiện chức năng receiver .

- Đối với Sending Thread thì khi task nhận message ở tại vị trí local thì sending thread chỉ gởi message xuống receiving queue thôi . Trong khi đó Receiving Thread ngoài nhiệm vụ nhận các message data từ các MsgPassDaemon khác nó còn nhận các message điều khiển của hệ thống.
- Các bước thực hiện của Message Passing Daemon như sau :



Các bước thực hiện của MsgPassDaemon

-Quá trình xử lý các command mà MsgPassDaemon nhận được như sau :



Quá trình xử lý command của MsgPassDaemon

- Trong đó khi nhận một data message của các MsgPassDaemon khác gởi tới thì Receiving Thread sẽ đưa message vào hàng nhận .Để các user task có thể truy xuất . Đối với Sending Thread khi đọc một message và xác định được host của nơi nhận message thì nếu cùng host với nơi gởi thì chỉ cần chuyển message đó vào receive queue thôi .
- Đối tượng hiện thực cho MsgPassDaemon như sau :

```
class ObjNetMngMsgPass {
        private :
                QueuMessage *queuSend ;          // sending queue
                QueuMessage *queuReceive;        // receiving queue
                ObjResource *objProcess;         // list of user task
                ObjResource *objHost ;           // list of MsgPassDaemon
        public :
                ObjNetMngMsgPass();
                ~ObjNetMngMsgPass();
                void ServerDoSend();
                void ServerDoReceive(int mainsocket);
};
```

-Chương trình MsgPassDaemon sẽ có một đối tượng kiểu ObjNetMngPass là một đối tượng toàn cục . Khi thực thi MsgPassDaemon tạo ra hai thread gọi thực thi hai quá trình send và receive như sau :

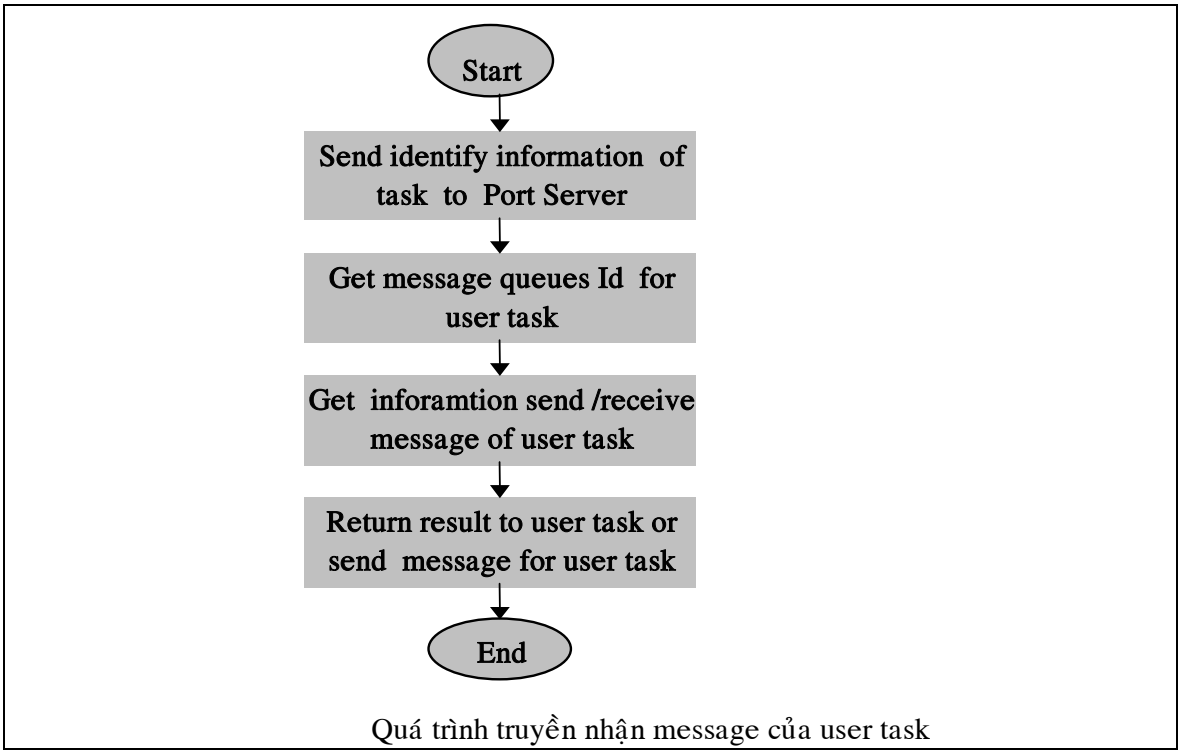
```
ObjNetMngMsgPass  *netMngMsgPass ;
thr_create(NULL,0,ThreadSend,(void*)NULL,THR_BOUND|THR_DETACHED,
                                                        (thread_t *) NULL );
```

```
thr_create(NULL,0,ThreadReceive,(void ) &main_socket , THR_BOUND |
THR_DETACHED , (thread_t *) NULL);
```

các thread này thực thi các đoạn chương trình sau :

```
void *ThreadSend(void *)
{
      netMngMsgPass->ServerDoSend() ;
       return (void *)NULL;
}
void *ThreadReceive(void *fd)
{
       int main_socket =(*( (int *)fd ) );
        netMngMsgPass->ServerDoReceive(main_socket);
       return (void *)NULL;
}
```

-Khi một user task muốn truyền message với một user task khác thông qua MsgPassDaemon thì hệ thống thực hiện quá trình thực hiện như sau :



Quá trình truyền nhận message của user task

-Các user task khi truyền nhận message thì chỉ cần biết được tên xác định của nơi nhận mà thôi . Hệ thống của chúng ta phải đảm bảo được message sẽ truyền đến đúng nơi cho task muốn nhận sẽ nhận được .

1.3.1.4 Truyền nhận message giữa các MsgPassDaemon:

- Các MsgPassDaemon dùng để truyền nhận message của user và trong hệ thống chúng đóng vai trò như cầu nối để truyền nhận message .Giữa các message chỉ có truyền nhận data message mà thôi . Cấu trúc message truyền nhận giữa các MsgPassDaemon là cấu trúc dữ liệu lưu trong message queues .

- Để quản lý hệ thống thì giữa các MsgPassDaemon và chương trình quản lý hệ thống có truyền nhận các command điều khiển .Có hai command như vậy :

◊ reset command : yêu cầu MsgPassDaemon thực hiện thao tác reset lại các queues .

◊ kill command : yêu cầu MsgPassDaemon loại bỏ các queue và kết thúc .

Các command dùng trong truyền message giữa các MsgPassDaemon và dùng quản lý hệ thống được trình bày ở bản đưới đây :

Bảng Các command trong truyền / nhận message của MsgPassDaemon

Command	Value	Description
MSG_DATA	100	command dùng truyền data
MSG_ACK	101	command dùng báo ack
MSG_RESET_PORT	102	command dùng báo cho MsgPassDaemon reset lại thông tin
MSG_KILL	103	command dùng báo cho MsgPassDaemon kết thúc thực thi

1.3.1.5. Thiết kế chương trình user task truyền / nhận message thông MsgPassDaemon :

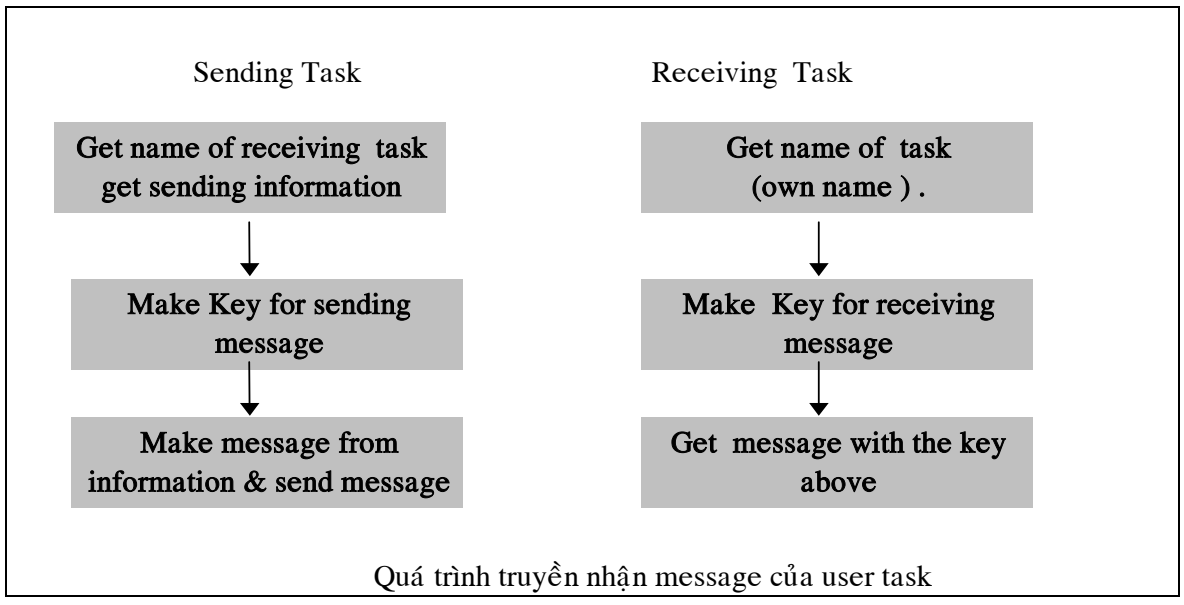
- Để user task có thể nhận gởi message thông MPD thì user task phải thực hiện các tao tác sau đây :

- initialize : dùng để báo cho hệ thống biết được rằng task này sẽ tham gia vào quá trình truyền nhận message thông qua MsgPassDaemon .
- access : cho phép user task thông qua các hàm mà tool cung cấp để truyền nhận message .
- close : dùng báo cho hệ thống biết rằng task sẽ kết thúc quá trình tham gia vào truyền nhận message thông qua MsgPassDaemon .

-User task sẽ có hai thao tác chính để truyền nhận message là :

- send messages .
- receive messages .

-Như đã nói ở trên việc xác định message nào là của task nào sẽ được xác định qua key của message . Mỗi task sẽ có một key dùng để nhận dạng được message của mình .Hệ thống thực thi quá trình truyền nhận message của user task như sau :



Quá trình truyền nhận message của user task

-Để cho phép user task truyền nhận message chúng ta dùng một đối tượng để quản lý các thao tác của user task . Đối tượng đó như sau :

```
class ObjUserMsgPassProc{
private :
      QueuMessage   *queuSend ;
      QueuMessage   *queuReceive ;
      char MY_NODE_NAME[20];
public :
      ObjUserMsgPassProc(char *nodename);
      ~ObjUserMsgPassProc();
      int msgSend(UserInfor *infor);
      int msgSend(UserInfor *infor ,char *nodename);
      int msgSend(UserInfor *infor ,char *nodename ,short timeout);
      int msgRecv(UserInfor *infor);
      int msgRecv(UserInfor *infor ,char *nodename);
      int msgRecv_T(UserInfor *infor,short timeout);
      int msgRecv_T(UserInfor *infor ,char *nodename,short timeout);
};
```

- Khi user task khai báo rằng sẽ truyền nhận message thông qua MsgPassDaemon thì hệ thống sẽ tạo ra một biến kiểu ObjUserMsgPassProc như sau :

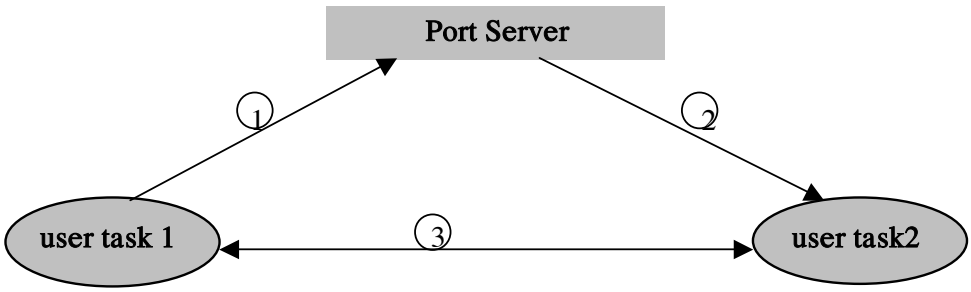
```
ObjUserMsgPassProc *objUserPassProc =new ObjUserMsgPassProc(nodename) ;
```

lúc bấy giờ hệ thống sẽ có các thông tin để truy xuất message queues .Khi truyền nhận message thì user task gọi các hàm và các hàm này sẽ gọi biến trên để xử lý .

1.3.2 IPC trực tiếp giữa hai process :

-Trong hệ thống của chúng ta cũng cho phép giữa hai process bất kỳ trong chương trình có thể trao đổi dữ liệu trực tiếp với nhau .Điều này cho phép vấn đề trao đổi giữa các process trở nên dễ dàng và thuận tiện . Hệ thống sẽ cung cấp một số chức năng để hai process có thể thực hiện trao đổi với nhau .

- Mô hình trao đổi trực tiếp như sau :

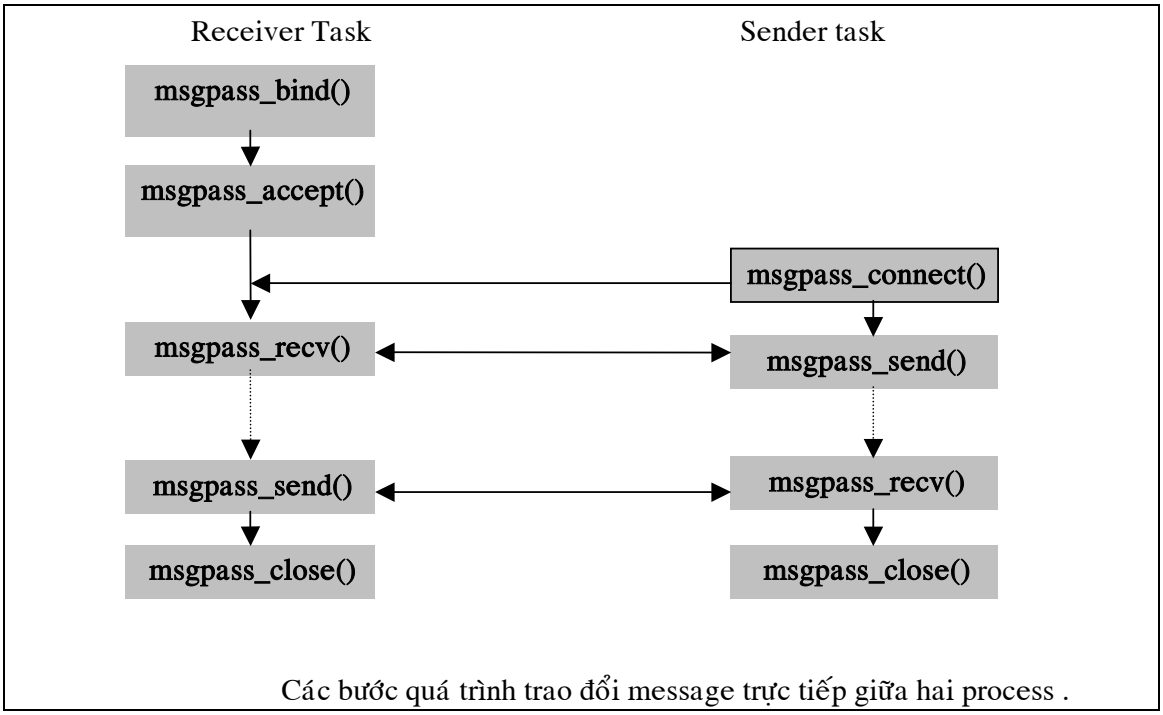


1.User task 1 báo vị trí :tên ,host ,port của mình về server quản lý .
2.User task 2 nhận thông tin về user task 1 từ server quản lý .
3.User task 2 connect tới user task1 và tiến hành trao đổi message .

-Để thực hiện trao đổi trực tiếp nhau thì cả nơi gởi và nơi nhận phải xác định rõ đối tượng của mình .Các bước tiến hành như sau :
-Việc thực hiện trao đổi message trực tiếp giữa hai task giống như trao đổi giữa client và server .Nơi nhận đóng vai trò server còn nơi gởi đóng vai trò client .Sau khi thiết lập cầu nối thì việc trao đổi message thông qua cầu nối đó. Nơi nhận có thể chấp nhận nhiều cầu nối của nhiều nơi gởi .Quá trình thực hiện thiết lập cầu nối cũng như trao đổi message giống như tạo socket .
- Khi một task là task nhận nghĩa là nó muốn nhận message từ task khác thì khi đó task phải thực hiện các thao tác như sau đây :

- thực hiện cài đặt các tác vụ cần thiết để tạo cầu nối .
- chấp nhận cầu nối và trao đổi dữ liệu .
- kết thúc một cầu nối và có thể chờ các cầu nối khác hay hủy bỏ
- hủy bỏ các cầu nối .

Quá trình trao đổi trực tiếp giữa hai process như sau :



Các bước quá trình trao đổi message trực tiếp giữa hai process .

1.3.3 Các vấn đề còn lại :

- Trong quá trình thiết kế và hiện thực các chương trình của tool cho phần message passing chúng ta đã hiện thực được việc trao đổi message giữa các task với nhau theo hai cơ chế tuy nhiên còn lại những vấn đề như sau :

Chưa giải quyết vần đề message passing theo group .
Chưa giải quyết vấn đề broadcast message .

- Các vấn đề này hiện tại chưa được hiện thực .Đó là các hướng phát triển sau này .

1.4 Các chương trình dùng setup và quản lý hệ thống :

-Trong hệ thống của chúng ta có sử dụng nhiều NSM Server ,MsgPass Daemon được quản lý bằng Port Server .Tuy nhiên cũng giống như bất kì một hệ thống nào công cụ của chúng ta phải thực hiện được các yêu cầu về setup và quản lý hệ thống :

Các yêu cầu cần thiết mà chúng ta phải đảm bảo:

- Tại mỗi máy chỉ có tối đa một chương trình NSM Server và một chương trình MsgPassDaemon .
- Hệ thống phải tự thiết lập theo cấu hình do user chỉ định.
- Hệ thống phải reset khi user tạo ra một workplace khác .
- Cho phép user có thể thêm bớt các host tham gia vào quá trình trong khi sử dụng tool .
- Hệ thống phải được loại bỏ khi user không sử dụng tool .

- Chúng ta sử dụng file để lưu trữ các thông tin sau :

◊ Thông tin các máy có chứa tool cùng tham gia .
◊ Thông tin các máy sẽ được load khi khởi động .
◊ Thông tin về Port Server,các NSM Server , MsgPass Daemon đang thực thi trên hệ thống .
◊ Thông tin về các tài nguyên dùng chung .

- Các thông tin này được dùng trong các trường hợp sau :

◊ Setup hệ thống .
◊ reset hệ thống .
◊ Thêm bớt các chương trình .
◊ Xem các thông tin .
◊ Loại bỏ tool .

- Sau đây chúng ta sẽ lần lược giới thiệu các tác vụ đó .

1.4.1 Kỹ thuật lập trình tránh nhiều bản copy của một chương trình trên máy :

- Trong hệ thống của chúng ta như chúng ta đã thiết kế tại một máy chỉ cho phép tối đa một NSM Server và một MsgPassDaemon . Do vậy chúng ta phải sử dụng kỹ thuật mutual exclusive cho chương trình Server khi nó bắt đầu thực thi . Một kỹ thuật đơn giản là sử dụng lock file để xác nhận mutual exclusive . Từng server với từng user khác nhau sẽ có một lock file khác nhau .

- Đoạn code hiện thực như sau : khi bắt đầu thực thi .

```
char *LOCKF = filename      //tên file được xác định duy nhất .
lf =open(LOCKF ,O_RDWR | O_CREAT ,0640) ;
if (lf < 0 ) {
        printf("Error in creating file lock \n");
        exit(1) ;
}
if (flock(lf ,LOCK_EX | LOCK_NB))
        exit(0);
khi Server kết thúc thì thực hiện đoạn code :
lf =open(LOCKF ,O_RDWR  ,0640) ;
if (lf < 0 ) {
        printf("Error in open file lock \n");
```

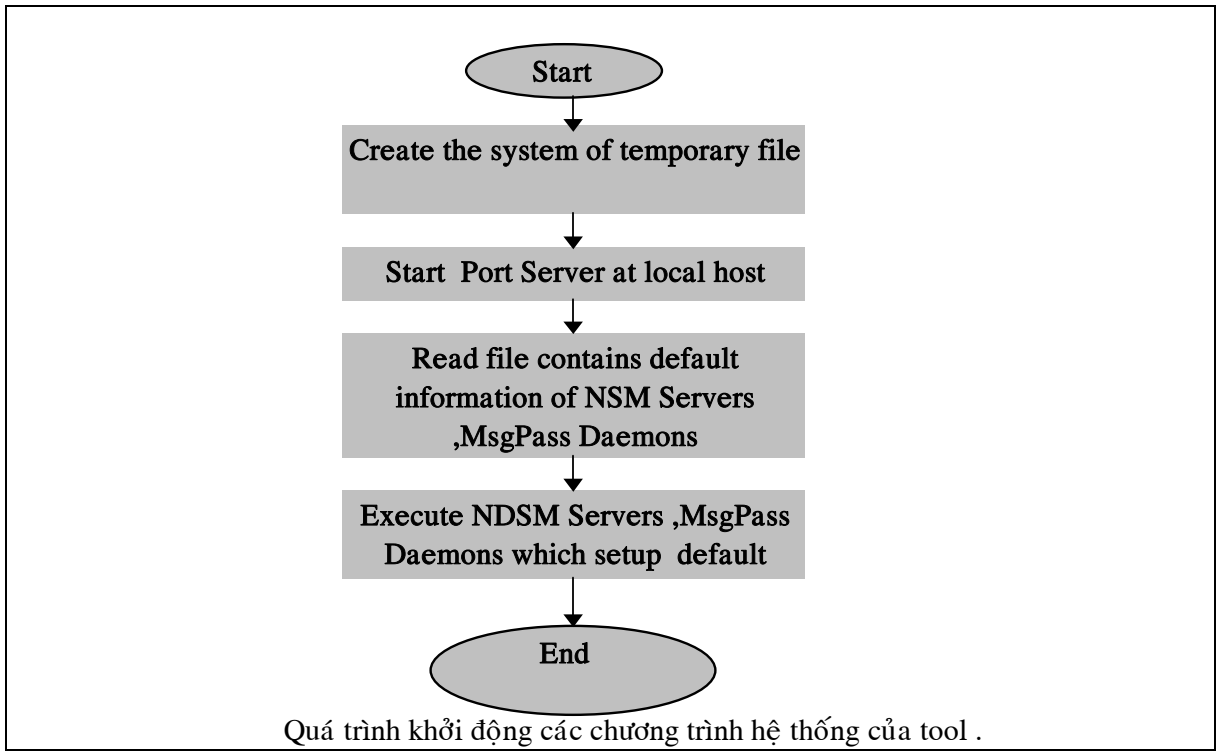
```
        exit(1) ;
}
flock(lf ,LOCK_UN | LOCK_NB) ;
        exit(0);
```

- Tất nhiên là có nhiều kỹ thuật khác nhưng kỹ thuật dùng file lock có nhiều yêu điểm đó là khi chương trình server bị hư hỏng hay khi máy reboot thì lock tự động giải phóng .

1.4.2 Quá trình khởi động hệ thống :

-Trong quá trình này chương trình chính sẽ đọc các file cấu hình của user và load các chương trình NSM Server ,MsgPassDaemon cần thiết theo yêu cầu của user .Sau khi khởi động thì user có thể thêm bớt các chương trình này .

- Quá trình khởi động hệ thống như sau :

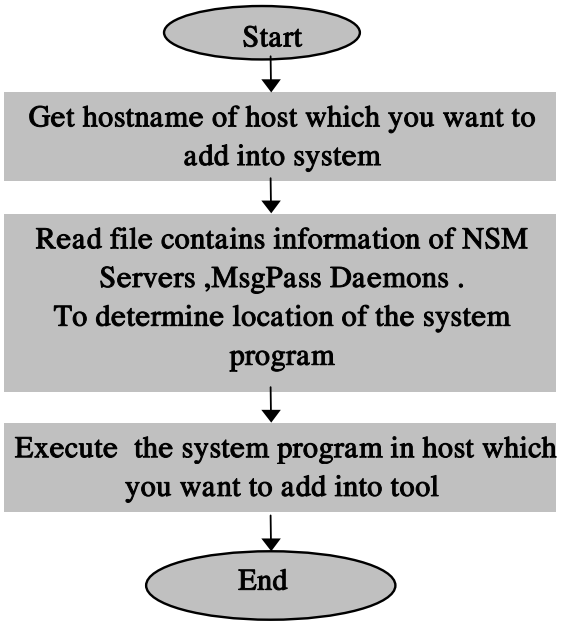


Quá trình khởi động các chương trình hệ thống của tool .

1.4.3 Quá trình thêm bớt các chương trình NSM Server hoặc MsgPassDaemon :

a . Thêm các chương trình NSM Server hoặc MsgPassDaemon :

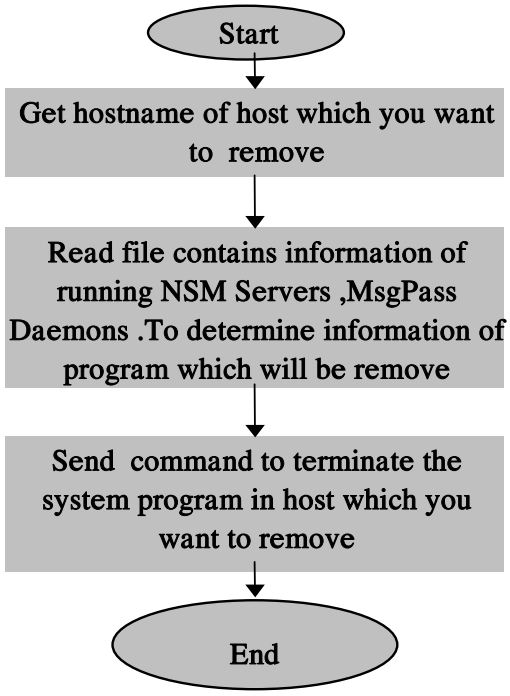
- Để có thể cho phép một host nào đó trong hệ thống chứa chương trình NSM Server hay MsgPassDaemon thì ta có thể thêm vào bất kì lúc nào sau khi thực thi theo cấu hình default . Hệ thống của chúng cho phép user có thể thêm vào NSM Server hay MsgPassDaemon vào bất kì lúc nào bằng các dùng command line .



Quá trình thêm một host vào trong tool

b. Bớt các chương trình NSM Server hoặc MsgPassDaemon :

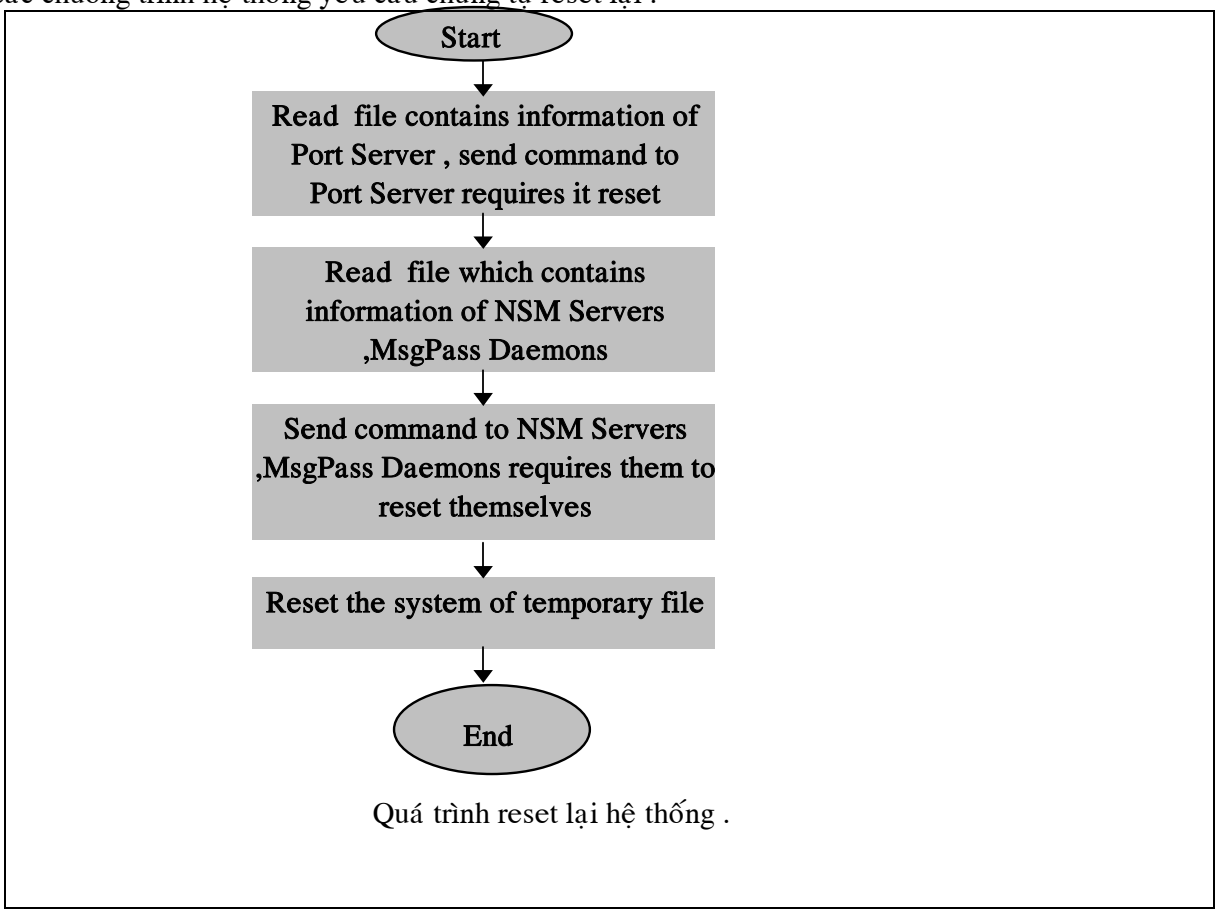
- Giống như quá trình thêm vào chúng ta cũng có quá trình loại bỏ bớt . Trình tự loại bỏ như sau :



Quá trình loại bỏ chương trình hệ thống của tool ở một máy nào đó .

1.4.4 Quá trình reset hệ thống :

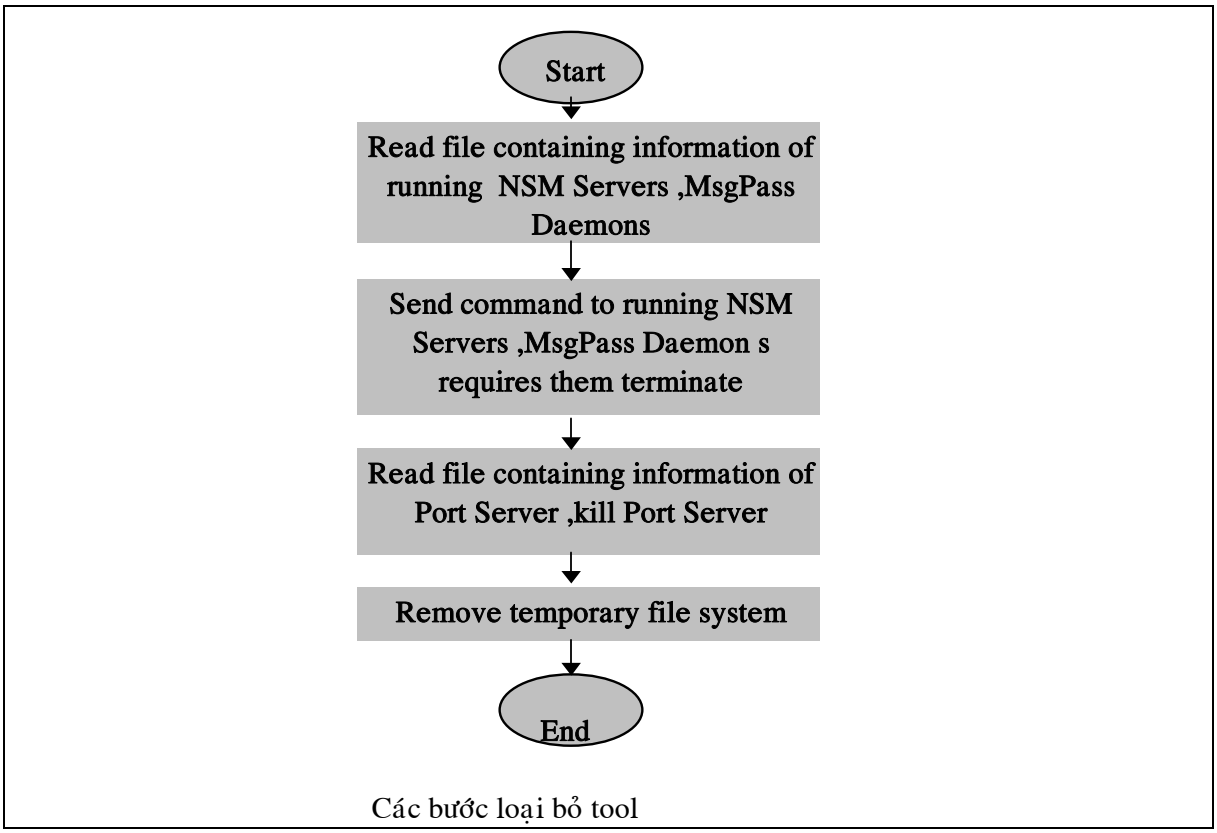
- Trong quá trình này dựa vào các thông tin trạng thái được lưu trữ chúng ta sẽ gởi các command tới các chương trình hệ thống yêu cầu chúng tự reset lại .



Quá trình reset lại hệ thống .

1.4.5 Quá trình loại bỏ tool :

- Trong quá trình này cũng dựa vào thông tin trạng thái của hệ thống chúng ta gởi các command đến các chương trình và khi chúng nhận được những thông tin này chúng sẽ loại bỏ các tài nguyên hệ thống mà nó quản lý đồng thời sẽ kết thúc quá trình thực thi .



Các bước loại bỏ tool

1.4.6 Quản lý các process của user :

-Để thuận tiện cho user cũng như cung cấp một môi trường lập trình tốt mà user có khả năng quản lý được các process của mình .Từ đó có thể biết được các process nào của mình đang thực thi ở máy nào giúp cho quản lý cũng như testing chương trình .

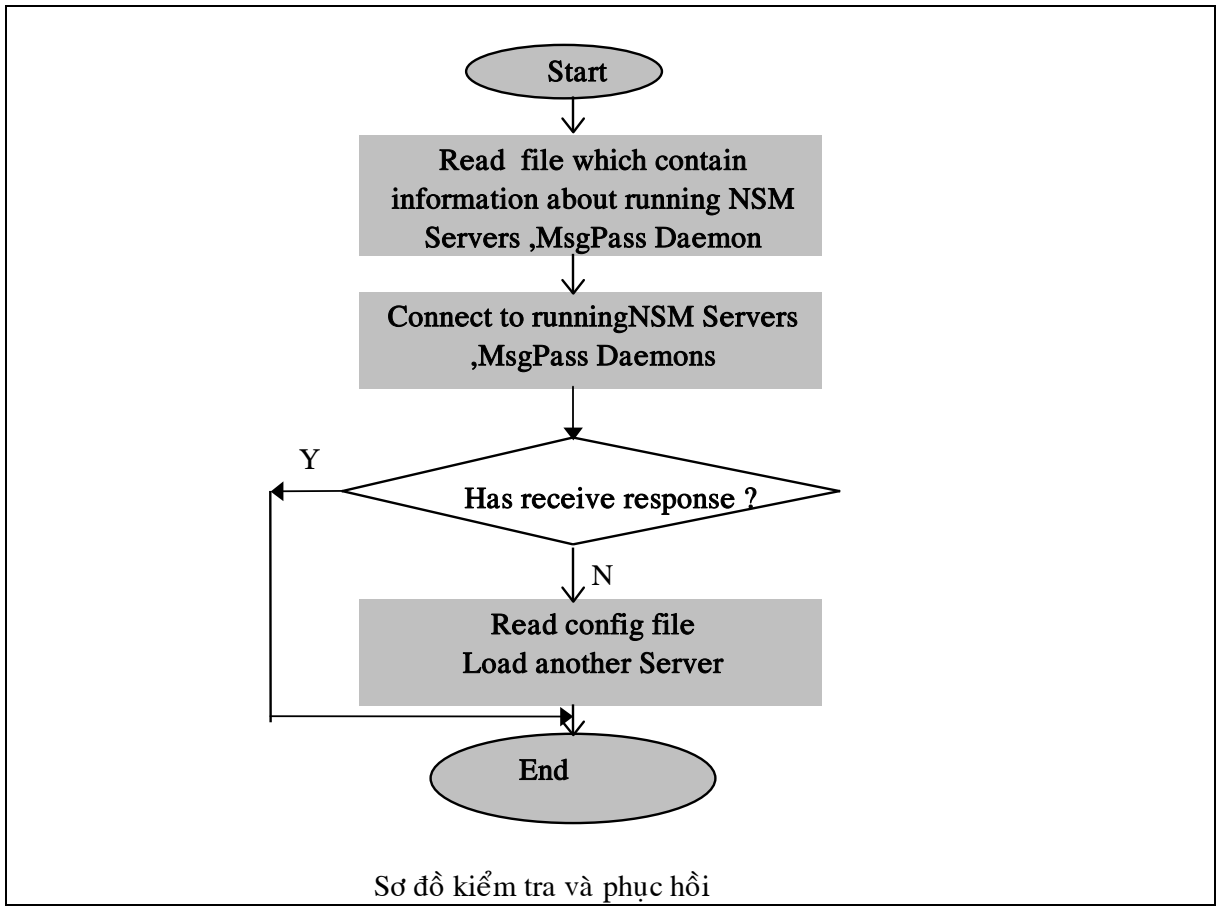
- Để thực hiện được điều đó thì chúng ta cung cấp cho user thêm các lệnh *ps* và *kill* của tool . Các lệnh này sẽ cho phép user có thể biết được các process của mình trên các máy cũng như có thể bỏ các process đó đi .

1.5 Faul tolerance and Recovery :

- Trong hệ thống của chúng ta các NSM Server các MsgPass Daemon vì một lý do nào đó có thể bị hư hỏng .Do vậy chúng ta phải có cơ chế kiểm tra và phục hồi khi biết các đối tượng này hư hỏng .Trong tool chúng ta khi thiết kế các PortServer ,NSM Server ,MsgPass Daemon ngoài các module có chức năng trao đổi message ,thao tác dữ liệu còn có chức năng giám sát .

- Đối với Port Server :chức năng này có vai trò kiểm tra các NSM Server ,MsgPass Daemon có trong danh sách lưu trữ có còn tồn tại hay không ? .Khi một NSM Server có trong danh sách mà không tồn tại thì có nghĩa là nó đã bị hư hỏng .Do đó phải có cơ chế phục hồi .
- Đối với NSM Server ,MsgPass Daemon : chức năng này có vai trò kiểm tra xem nó có còn liên lạc được với Port Server hay không .Một khi nó không còn liên lạc được với Port Server thì có nghĩa là nó không thể nào tham gia vào các quá trình của công cụ nữa vì nó phải tự hủy diệt .

- Quá trình giám sát và phục hồi như sau :

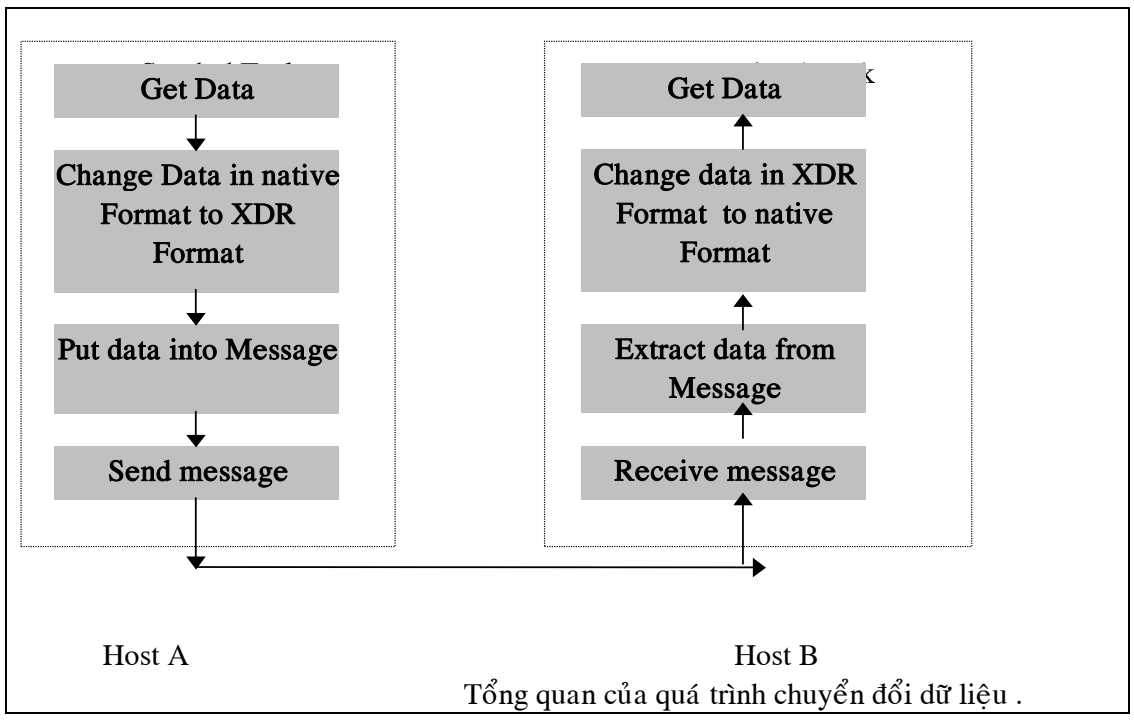


Sơ đồ kiểm tra và phục hồi

- Quá trình này được một thread của Port Sever gọi thực thi sau một khoảng thời gian cố định .

2.Các hàm chuyển đổi data :

2.1 Sơ đồ tổng quát quá trình chuyển đổi data :



Tổng quan của quá trình chuyển đổi dữ liệu .

- Ở trên đây là quá trình chuyển đổi dữ liệu tổng quan Như chúng ta đã phân tích quá trình chuyển đổi data của chúng ta là quá trình đối xứng .Dữ liệu sẽ được chuyển đổi trước khi truyền đi và tại nơi nhận dữ liệu sẽ được đổi lại .

-Việc sử dụng các hàm dùng để chuyển đổi dữ liệu hoàn toàn phụ thuộc vào user . Khi nào cần thiết chuyển đổi thì user sẽ chuyển đổi .Tất nhiên là điều này có thể bất tiện cho user trong vấn đề xác định có chuyển đổi hay không như nó cho phép uyển chuyển trong việc chuyển đổi dữ liệu . Chúng ta sẽ chuyển đổi khi cần thiết ,tránh lãng phí trong khi chuyển đổi không cần thiết .

2.2 Quá trình chuyển đổi dữ liệu trong tool :

2.2.1 Các kiểu dữ liệu được chuyển đổi :

- Trong tool ta định nghĩa một số kiểu dữ liệu như sau :

Bảng các kiểu dữ liệu trong tool :

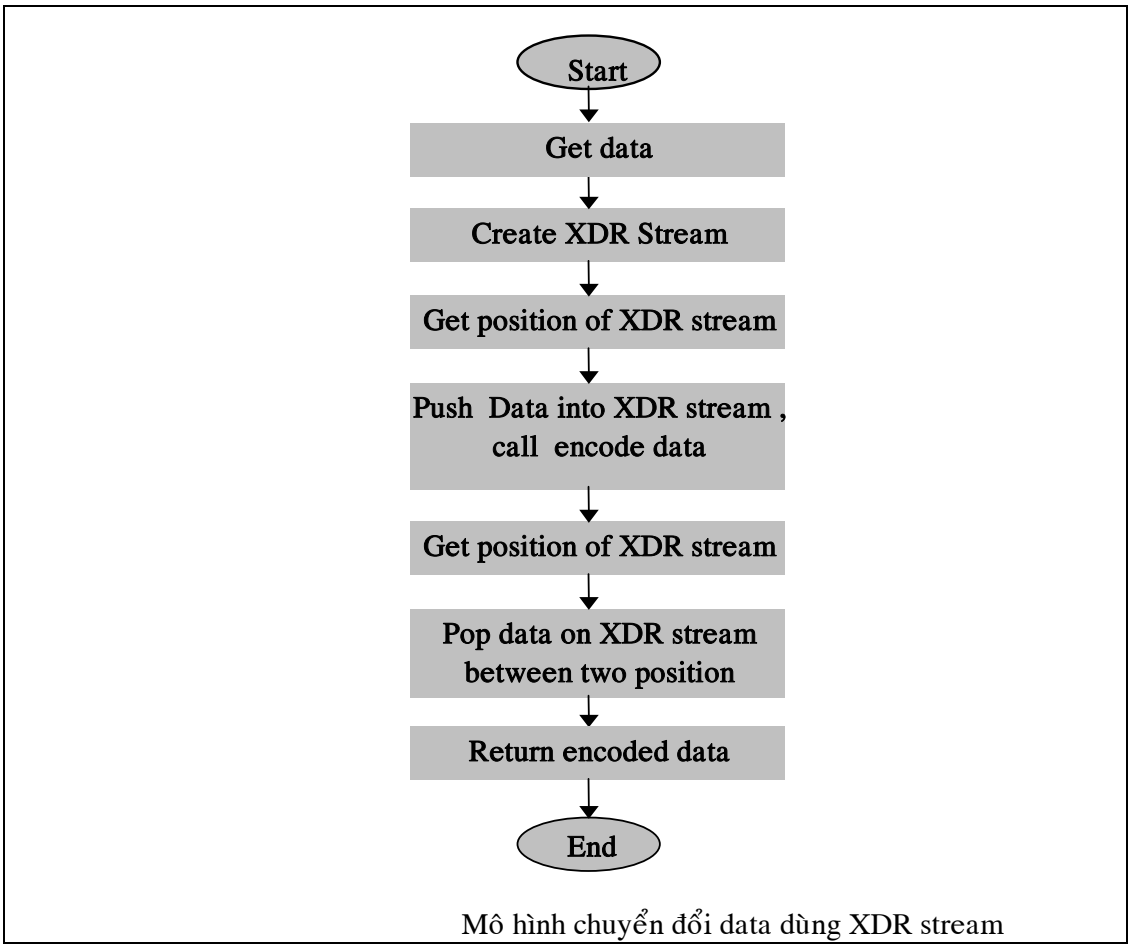
Name	Value	Description
DPPT_BOOL	1	Kiểu dữ liệu bool (thực chất là số int)
DPPT_CHAR	2	Kiểu dữ liệu char
DPPT_DOUBLE	3	Kiểu double
DPPT_FLOAT	5	Kiểu float
DPPT_INT	6	Kiểu int
DPPT_LONG	7	Kiểu long
DPPT_LONGLONG	8	Kiểu long long
DPPT_QUADRUPLE	9	Kiểu Qudruple
DPPT_SHORT	10	Kiểu short
DPPT_U_HYPER	12	Kiểu unsigned hyper (u_longlong_t)
DPPT_U_INT	13	Kiểu unsigned int
DPPT_U_LONG	14	Kiểu unsigned long
DPPT_U_LONGLONG	15	Kiểu u_longlong_t
DPPT_U_SHORT	16	Kiểu unsigned short
DPPT_HYPER	17	Kiểu hyper

2 .2 .2 Các hàm chuyển đổi :

- Trong tool sử dụng hai loại hàm chuyển đổi cơ bản :

  - Hàm chuyển đổi đơn giản : chuyển đổi các phần tử đơn giản như số int ,float ...
  - Hàm chuyển đổi theo dãy : các hàm này chuyển đổi theo dãy các phần tử cơ bản như dãy số int ,double ..

- Sơ đồ của quá trình chuyển đổi như sau :



Mô hình chuyển đổi data dùng XDR stream

2.3.Các vấn đề còn lại :

-Các hàm chuyển đổi được hiện thực nhiều tuy nhiền còn một số vấn đề sau :

- Chưa hiện thực chuyển đổi các dữ liệu có kiể một byte .
-Chưa thực hiện chuyển đổi string , kiểu structure ...
- Các kiểu phức tạp .

- Tuy nhiên với mô hình ứng dụng của tool thì các hàm trên cũng là khá đầy đủ .

3. Các thành phần khác :

- Ngoài các phần trên để có thiết kế được chúng ta cũng phải thiết kế rất nhiều các function ,object khác để phụ trợ cho việc thiết kế các chương trình của chúng ta .

PHẦN 4

THƯ VIỆN CHO NGƯỜI LẬP TRÌNH

NỘI DUNG :

- PHƯƠNG PHÁP CÀI ĐẶT CHƯƠNG TRÌNH DPPT .
- CÁC LỆNH CỦA TOOL .
- THƯ VIỆN CÁC HÀM CHO USER (DPPT LIBRARY) .

1. SETUP TOOL VÀ CÁC LỆNH HỆ THỐNG CỦA TOOL :

1.1 Setup tool :

- Để tool có thể thực thi thi chúng ta phải setup tool theo dúng yêu cầu :

-Cấu trúc chương trình của tool như sau trong thư mục :

```
dppt_version1
    ├─bin        // chứa các file thực thi
    ├─include    // chứa các file include
    └─lib        // chứa các thư viện
```

-Quá trình setup tool :

- Copy hệ thống tool vào account của người sử dụng hay vào hệ thống file của máy mà muốn thực thi tool .
- User muốn sử dụng :

◊ Tạo thư mục có tên *.DPPT_TEMP_DIRECTORY* tại *HOME directory* .

◊ Set các biến môi trường :

⇒ DPPT_MAINPATH : chỉ đường dẫn đến thư mục dppt_version1 vd : DPPT_MAINPATH =$HOME/dppt_version1

⇒ DPPT_BINPATH =$DPPT_MAINPATH/bin

⇒ DPPT_INCLUDEPATH=$DPPT_INCLUDEPATH

⇒ DPPT_LIBPATH=$DPPT_LIBPATH

◊ Tạo các yêu cầu về lệnh rsh và rcp giữa các máy để có thể thực thi các lệnh này được . Đồng thời cũng tạo các yêu cầu về cho phép truy xuất từ xa .

◊ Sửa đổi nội dung các file DPPT_SHARESERVER.CONFIG và DPPT_MSGPASS.CONFIG chỉ đến các chương trình ở các máy mà có thể cùng thực thi .

◊ Sửa đổi nội dung các file DPPT_SHARESERVER.CONFIG_DEFAUL và DPPT_MSGPASS.CONFIG_DEFALU chỉ đến các chương trình ở các máy mà có thể cùng thực thi khi tool thực thi .

◊ Khi sửa đổi các file trên phải đảm bảo dúng syntax như sau :

hostname[hostip]:programinthishost:

lưu ý không có khoảng trắng trước *hostname*

- Khi đó có thể thực thi tool bằng command # *DPPT*

1.2 Các lệnh hệ thống của tool :

-Để thực thi hệ thống có cho phép một số lệnh thao tác bằng command line dưới đây :

◊ **dppt_addMsgD hostname** :

- Lệnh này sẽ đọc trong file cấu hình của tool về message passing xác định vị trí các chương trình hệ thống của máy thêm vào .Nếu máy đó có trong file thì gọi thực thi các hàm chương trình hệ thống message passing trên máy đó . Khi đó máy đó có thể tham gia vào quá trình message passing .

◊ **dppt_addShareD hostname** :

- Lệnh này sẽ đọc trong file cấu hình của tool về NSM Server xác định vị trí các chương trình hệ thống của máy thêm vào .Nếu máy đó có trong file thì gọi thực thi các hàm chương trình hệ thống NSM Server trên máy đó . Khi đó có một thêm NSM Server trên máy đó .

◊ **dppt_removeMsgD hostname :**

-Lệnh này sẽ loại bỏ máy *hostname* ra khỏi các máy tham gia vào message passing .

◊ **dppt_removeShareD hostname :**

-Lệnh này sẽ loại bỏ NSM Server trên máy có tên *hostname* .

◊ **dppt_resetMsgD hostname:**

-Lệnh này sẽ reset MsgPassDaemon đang thực thi trên hostname .

◊ **dppt_resetShareD hostname :**

- Lệnh này reset NSM Server trên hostname .

◊ **dppt_resetAllMsgD :**

-Lệnh này sẽ reset toàn bộ các MsgPassDaemon đang thực thi trên hệ thống .

◊ **dppt_resetAllShareD :**

- Lệnh này reset toàn bộ các NSM Server trên hệ thống .

◊ **dppt_killAllMsgD :**

-Lệnh này kill tất các các MsgPassDaemon đang thực thi trong hệ thống .

◊ **dppt_killAllShareD :**

-Lệnh này sẽ kill tất cả các NSM Server đang thực thi trong hệ thống .

◊ **dppt_chechSystem :**

- Lệnh kiểm tra các chương trình hệ thống đang thực thi có còn tồn tại hay không . Nếu không thì load lại .

◊ **dppt_ps [hostname] :**

-Lệnh này sẽ liệt kê các process của user tại local hay tại hostname .

◊ **dppt_kill [hostname] pid :**

- Lệnh này kill một process có process id là pid tại local hoặc tai hostname.

2. CÁC HÀM CUNG CẤP CHO USER TASK :

-Các hàm này được khai báo trong DPPT.h

2.1 Các hàm thao tác trên NDSM Server :

- **int dppt_shareserver_start() :**

- Hàm này dùng để khởi động hệ thống cho user task để có thể thao tác với dữ liệu dùng chung trên các NSM Server .Nếu user task không khởi động hàm này thì quá trình truy xuất NSM Server sẽ không được .

- **int dppt_shareserver_end():**

- Hàm này dùng báo cho hệ thống biết user task không còn tham gia vào quá trình truy xuất dữ liệu trên NSM Server và hệ thống có thể loại bỏ các biến hệ thống mà dùng cho việc truy xuất NSM Server .

- **unsigned int dppt_getpid() :**

- Hàm này trả về một số unsigned int như là một process id của một task trong hệ thống của chúng ta . Giá trị hàm này hiện tại chưa được sử dụng .

- **int dppt_ServerAlive(short timeout):**

- Hàm này kiểm tra xem có tồn tại NSM Server hay không ,Nếu quá thời gian **timeout** giây mà không có tín hiệu từ NSM Server thì xem như không có server nào tồn tại .

- **int dppt_initVar(char *name_var,int size,short timeout):**

- Hàm này dùng thiết lập một biến toàn cục trên NSM Server .Chuỗi name_var chỉ đến tên biến , giá trị size chỉ kích thước biến còn timeout thì dùng cho việc timeout . Biến được thiết lập ở đây là biến không thể chia sẻ nhỏ hơn .

- Mã trả về :
  - ◊ Thành công :khi initialize thành công thì trả về số bytes mà thiết lập được .
  - ◊ Thất bại :

⇒ MALLOC_MEM_ERR : malloc bị lỗi .
⇒ TASK_FAIL : quá trình initialization bị lỗi vì một lý do nào đó .

- **int dppt_readVar(char *name_var ,char *rectdata ,int dataSize,short timeout):**

- Hàm này dùng đọc giá trị một biến toàn cục trên NSM Server .Chuỗi name_var chỉ đến tên biến , giá trị size chỉ kích thước biến còn timeout thì dùng cho việc timeout . Biến được đọc ở đây được thiết lập bởi hàm **dppt_initVar()** đây là biến không thể chia sẻ nhỏ hơn . Hàm này cho phép đọc mà` không cần quan tâm đến biến đó có bị lock hay không ,hoặc có task nào ghi vào biến đó hay không . Nói các khác đọc mà không đảm bảo . Nếu sau khoảng timeout mà không đọc được thì sẽ timeout và báo lỗi .
- Mã trả về :
  ◊ Thành công : trả về số bytes yêu cầu đọc .
  ◊ Thất bại :
    ⇒ RESOURCE_NOT_FOUND : không tìm thấy biến có tên như vậy .
    ⇒ TASK_FAIL : tác vụ không thực hiện được vì các lý do khác (như biến này chưa được ghi giá trị) .

- **int dppt_readVar_S(char *name_var ,char *rectdata ,int dataSize,short timeout):**

- Hàm này dùng đọc giá trị một biến toàn cục trên NSM Server .Chuỗi name_var chỉ đến tên biến , giá trị size chỉ kích thước biến còn timeout thì dùng cho việc timeout . Biến được đọc ở đây được thiết lập bởi hàm *dppt_initVar()* , đây là biến không thể chia sẻ nhỏ hơn . Hàm này đọc biến với cơ chế đảm bảo . Khi biến đó bị một process khác lock hoặc có process nào ghi đang ghi thì nó sẽ chờ cho đến khi biến không bị lock hoặc không có process nào ghi vào . Nếu sau khoảng timeout mà không đọc được thì sẽ timeout và báo lỗi .
- Mã trả về của hàm này giống như hàm *dppt_readVar() .*

- **int dppt_writeVar(char *name_var ,char *inputdata ,int dataSize,short timeout):**

- Hàm này dùng ghi giá trị một biến toàn cục trên NSM Server .Chuỗi name_var chỉ đến tên biến , giá trị size chỉ kích thước biến còn timeout thì dùng cho việc timeout . Biến được ghi ở đây được thiết lập bởi hàm **dppt_initVar()** đây là biến không thể chia sẻ nhỏ hơn . Hàm này cho phép ghi mà không cần quan tâm đến biến đó có bị lock hay không ,hoặc có task nào đọc , ghi vào biến đó hay không . Nói các khác ghi mà không đảm bảo . Nếu sau khoảng timeout mà không ghi được thì sẽ timeout và báo lỗi .
- Mã trả về :
  ◊ Thành công : trả về số bytes được ghi .
  ◊ Thất bại :
    ⇒ RESOURCE_NOT_FOUND : không tìm thấy biến có tên như vậy .
    ⇒ TASK_FAIL : tác vụ không thực hiện được vì các lý do khác (như biến này chưa được ghi giá trị) .

- **int dppt_writeVar_S(char *name_var ,char *inputdata ,int dataSize,short timeout):**

- Hàm này dùng ghi giá trị một biến toàn cục trên NSM Server .Chuỗi *name_var* chỉ đến tên biến , giá trị *dataSize* chỉ kích thước biến ,còn giá trị *timeout* dùng cho việc timeout . Biến được ghi ở đây được thiết lập bởi hàm *dppt_initVar()* , đây là biến không thể chia sẻ nhỏ hơn . Hàm này đọc biến với cơ chế đảm bảo . Khi biến đó bị một process khác lock hoặc có process nào đang ghi hoặc đang đọc thì nó sẽ chờ cho đến khi biến không bị lock hoặc không có process nào đọc hoặc ghi vào . Nếu sau khoảng timeout mà không đọc được thì sẽ timeout và báo lỗi .
- Mã trả về giống như trong hàm *dppt_writeVar_S()* .

- **int dppt_removeVar(char *name_var ,short timeout):**

- Hàm này dùng loại bỏ một biến toàn cục trên NSM Server .Chuỗi *name_var* chỉ đến tên biến ,*timeout* thì dùng cho việc timeout . Biến được loại bỏ ở đây được thiết lập bởi hàm *dppt_initVar()*. Hàm này loại bỏ biến với cơ chế đảm bảo . Khi biến đó bị một process khác lock hoặc có process nào đang ghi hay đọc thì nó sẽ chờ cho đến khi biến không bị lock hoặc không có process nào đọc hay ghi vào .Sau đó sẽ loại bỏ biến. Nếu sau khoảng timeout mà không đọc được thì sẽ timeout và báo lỗi .
- Mã trả về :
  - ◊ Thành công : trả về giá trị TASK_SUCCESSFUL .
  - ◊ Thất bại :
    - ⇒ RESOURCE_NOT_FOUND : không tìm thấy biết có tên như vậy .
    - ⇒ TASK_FAIL : tác vụ không thực hiện được vì các lý do khác (như biến này chưa được ghi giá trị) .

- **int dppt_removeAllVar(char *name_var,short timeout):**

-Hàm này tương tự như hàm **dppt_removeVar()** nhưng hàm này loại bỏ tất cả các biến chung trên các NSM Server .
- Mã trả về :
  - ◊ Thành công : trả về giá trị TASK_SUCCESSFUL .
  - ◊ Thất bại :
    - ⇒ RESOURCE_NOT_FOUND : không tìm thấy biết có tên như vậy .
    - ⇒ TASK_FAIL : tác vụ không thực hiện được vì các lý do khác .

- **int dppt_readLockVar(char *name_var ,short timeout):**

- Hàm này dùng khóa một biến trên NSM Server với đặc tính read lock . Khi một process khóa biến read lock thì các process khác nếu sử dụng cơ chế đọc ghi đảm bảo thì chỉ được đọc . Nếu biến đã bị lock write thì nó sẽ chờ cho đến khi biến không bị lock write rồi sẽ tiến hành khóa .Giá trị timeout dùng chỉ thời gian timeout .
- Mã trả về :
  - ◊ Thành công : trả về giá trị TASK_SUCCESSFUL .
  - ◊ Thất bại :
    - ⇒ RESOURCE_NOT_FOUND : không tìm thấy biết có tên như vậy .
    - ⇒ TASK_FAIL : tác vụ không thực hiện được vì các lý do khác .

- **int dppt_tryReadLockVar(char *name_var ,short timeout):**

- Hàm này giống như **dppt_readLockVar()** chỉ khác là nếu biến đã bị write lock thì nó sẽ không chờ mà trở về ,khi đó xem như khóa biến chỉ đọc thất bại .
- Mã trả về giống như hàm *dppt_readLockVar()* .

- **int dppt_writeLockVar(char *name_var,short timeout):**

- Hàm này dùng khóa một biến trên NSM Server với đặc tính write lock . Khi một process khóa biến write lock thì các process khác nếu sử dụng cơ chế đọc ghi đảm bảo thì phải chờ cho đến khi không khóa . Nếu biến đã bị lock write hay lock read thì nó sẽ chờ cho đến khi biến không bị lock rồi sẽ tiến hành khóa .Giá trị timeout dùng chỉ thời gian timeout .
- Mã trả về giống như hàm *dppt_readLockVar()* .

- **int dppt_tryWriteLockVar(char *name_var,short timeout):**

- Hàm này giống như ***dppt_writeLockVar()*** chỉ khác là nếu biến đã bị write lock hoặc read lock thì nó sẽ không chờ mà trở về ,khi đó xem như khóa biến với tính chất write lock thất bại .
- Mã trả về giống như hàm *dppt_readLockVar()* .

- **int dppt_unlockLockVar(char *name_var ,short timeout):**

- Hàm này dùng giải phóng một khóa của biến . Khoá của biến là read lock hoặc write lock sẽ bị giải phóng .
- Mã trả về giống như hàm *dppt_readLockVar()* .

- **int dppt_initMatrix(char *matrix_name ,int nRow ,int nCol,int elementSize ,short timeout):**

- Hàm này dùng để initialize một matrix cấp NxM trên NSM Server .Chuỗi vertor_name chỉ tên của matrix ,giá trị nRow chỉ số hàng , nCol chỉ số cột ,elementSize chỉ kích thước của thành phần matrix . Nói cách khác hàm này initial một ma trận **matrix_name[nRow][nCol] of elementSize .** Giá trị timeout giống như các hàm trên .
- Mã trả về :
  - ◊ Thành công :khi initialize thành công thì trả về giá trị TASK_SUCCESSFUL .
  - ◊ Thất bại :
    - ⇒ MALLOC_MEM_ERR : malloc bị lỗi .
    - ⇒ TASK_FAIL : quá trình initialization bị lỗi vì một lý do nào đó .

- **int dppt_readItemMatrix(char *matrix_name , char *rectdata,int dataSize,short int sRow,short int sCol,int timeout):**

- Hàm này đọc phần tử *matrix_name[sRow][sCol]* có kích thước *dataSize bytes* và trả kết quả về trong vị trí chỉ bởi pointer *rectdata* . Kích thước của vùng nhớ rectdata phải bằng dataSize và kích thước phần tử của matrix có tên matrix_name cũng phải bằng dataSize .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_readRowMatrix(char *matrix_name ,char *rectdata,int dataSize,short int sRow ,short int sCol ,short int nElement ,int timeout):**

- Hàm này đọc một hàng của matrix có tên *matrix_name*.Vị trí phần tử bắt đầu đọc là *matrix_name[sRow][sCol]* ,số phần tử đọc là *nElement* và trả kết quả về trong vị trí chỉ bởi pointer *rectdata* . Kích thước của vùng nhớ rectdata phải bằng *dataSize* và kích thước của toàn bộ hàng đọc về phải bằng *dataSize* .Giá trị timeout dùng cho việc timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_readColMatrix(char *matrix_name ,char *rectdata ,int dataSize,short int sRow ,short int sCol ,short int nElement ,short timeout):**

- Hàm này đọc một cột của matrix có tên *matrix_name*.Vị trí phần tử bắt đầu đọc là *matrix_name[sRow][sCol]* ,số phần tử đọc là *nElement* và trả kết quả về trong vị trí chỉ bởi pointer *rectdata* . Kích thước của vùng nhớ rectdata phải bằng *dataSize* và kích thước của toàn bộ cột đọc về phải bằng *dataSize* .Giá trị timeout dùng cho việc timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_readSubMatrix(char *matrix_name ,char *rectdata ,int dataSize,short int sRow ,short int sCol ,short int eRow ,short int eCol ,short timeout):**

- Hàm này đọc một submatrix của matrix có tên *matrix_name*.Vị trí phần tử bắt đầu đọc là *matrix_name[sRow][sCol]* , vị trí phần tử kết thúc là *matrix_name[eRow][eCol]* và trả kết quả về trong vị trí chỉ bởi pointer *rectdata* . Kích thước của vùng nhớ rectdata phải bằng *dataSize* và kích thước của submatrix đọc về phải bằng *dataSize* .Giá trị timeout dùng cho việc timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_writeItemMatrix(char *matrix_name , char *inputdata,int dataSize,short int sRow,short int sCol,short timeout):**

- Hàm này ghi giá trị vào phần tử *matrix_name[sRow][sCol]* có kích thước *dataSize bytes* của matrix có tên matrix_name .Dữ liệu dùng để ghi chứa trong vị trí được chỉ bởi pointer input*data* . Kích thước của vùng nhớ *inputdata* phải bằng dataSize và kích thước phần tử của matrix có tên matrix_name cũng phải bằng dataSize . Giá trị timeout dùng chỉ thời gian timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_writeRowMatrix(char *matrix_name ,char *inputdata ,int dataSize,short int sRow ,short int sCol ,short int nElement ,short timeout):**

- Hàm này ghi vào một hàng của matrix có tên *matrix_name*.Vị trí phần tử bắt đầu ghi là *matrix_name[sRow][sCol]* ,số phần tử ghi là *nElement* và dữ liệu dùng để ghi chứa trong trong vị trí chỉ bởi pointer *inputdata* . Kích thước của vùng nhớ *input*data phải bằng *dataSize bytes* và kích thước của toàn bộ hàng được ghi phải bằng *dataSize bytes* .Giá trị timeout dùng cho việc timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_writeColMatrix(char *matrix_name ,char *inputdata,int dataSize ,short int sRow ,short int sCol ,short int nElement,short timeout):**

- Hàm này ghi vào một cột của matrix có tên *matrix_name*.Vị trí phần tử bắt đầu ghi là *matrix_name[sRow][sCol]* ,số phần tử ghi là *nElement* và dữ liệu dùng để ghi chứa trong trong vị trí chỉ bởi pointer *inputdata* . Kích thước của vùng nhớ *input*data phải bằng *dataSize bytes* và kích thước của toàn bộ cột được ghi phải bằng *dataSize bytes* .Giá trị timeout dùng cho việc timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_writeSubMatrix(char *matrix_name ,char *inputdata,int dataSize ,short int sRow ,short int sCol ,short int eRow ,short int eCol,short timeout):**

- Hàm này ghi vào một submatrix của matrix có tên *matrix_name*.Vị trí phần tử bắt đầu ghi là *matrix_name[sRow][sCol]* ,vị trí phần tử ghi là *matrix_name[sRow][eCol]* và dữ liệu dùng để ghi chứa trong trong vị trí chỉ bởi pointer *inputdata* . Kích thước của vùng nhớ *inputdata* phải bằng *dataSize bytes* và kích thước của toàn bộ submatrix được ghi phải bằng *dataSize bytes* .Giá trị timeout dùng cho việc timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_removeMatrix(char *matrix_name,short timeout):**

- Hàm này dùng loại bỏ một matrix có tên được chỉ bởi *matrix_name* trên NSM Server với thời gian timeout là *timeout* giây .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_removeAllMatrix(short timeout):**

- Hàm này dùng loại bỏ toàn bộ các matrix trên NSM Server với thời gian timeout là *timeout* giây .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_initVector(char *vector_name ,int nElement ,int elementSize,short timeout):**

- Hàm này dùng để initialize một vector trên NSM Server .Chuỗi *vertor_name* chỉ tên của vector ,giá trị nElement chỉ số phần tử của vector ,elementSize chỉ kích thước của thành phần vector . Nói cách khác hàm này initialize một vector *vector_name[nRow][nCol] of elementSize* . Giá trị timeout giống như các hàm trên .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_readVectorItem(char *vector_name ,char *rectdata ,int dataSize,short int Index,short timeout):**

- Hàm này đọc phần tử *vector_name[Index]* có kích thước *dataSize bytes* và trả kết quả về trong vị trí chỉ bởi pointer *rectdata* . Kích thước của vùng nhớ *rectdata* phải bằng *dataSize bytes* và kích thước phần tử của vector có tên *vector_name* cũng phải bằng *dataSize byets* . giá trị *timeout* dùng chỉ thời gian timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_readSubVector(char *vector_name ,char *rectdata ,int dataSize,short int Index ,short int nElement,short timeout):**

- Hàm này đọc giá trị một subvector của vector có tên *vector_name*.Vị trí phần tử bắt đầu đọc là *vector_name[Index]* ,số phần tử đọc là *nElement* và dữ liệu đọc được chứa vào trong vị trí chỉ bởi pointer *rectdata* . Kích thước của vùng nhớ *rectdata* phải bằng *dataSize bytes* và kích thước của toàn bộ subvector đọc được phải bằng *dataSize bytes* .Giá trị *timeout* dùng cho việc timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_writeVectorItem(char *vector_name ,char *inputdata ,int dataSize,short int Index,short timeout):**

- Hàm này ghi giá trị vào phần tử *vector_name[Index]* có kích thước *dataSize bytes* và giá trị dữ liệu dùn để ghi chứa trong vị trí chỉ bởi pointer *inputdata* . Kích thước của vùng nhớ *inputdata* phải bằng *dataSize bytes* và kích thước phần tử được ghi của vector có tên *vector_name* cũng phải bằng *dataSize byets* . Giá trị *timeout* dùng chỉ thời gian timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_writeSubVector(char *vector_name ,char *inputdata ,int dataSize,short int Index ,short int nElement,short timeout):**

- Hàm này ghi vào một subvector của vector có tên *vector_name*.Vị trí phần tử bắt đầu ghi là *vector_name[Index]* ,số phần tử ghi là *nElement* và dữ liệu dùng để ghi chứa trong trong vị trí chỉ bởi pointer *inputdata* . Kích thước của vùng nhớ *inputdata* phải bằng *dataSize bytes* và kích thước của toàn bộ subvector được ghi phải bằng *dataSize bytes* .Giá trị *timeout* dùng cho việc timeout .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_removeVector(char *vector_name,short timeout):**

- Hàm này dùng loại bỏ một matrix có tên được chỉ bởi *matrix_name* trên NSM Server với thời gian timeout là *timeout* giây .
- Mã trả về giống như hàm *dppt_initMatrix()* .

- **int dppt_removeAllVector(short timeout):**

- Hàm này dùng loại bỏ toàn bộ các vector trên NSM Server với thời gian timeout là *timeout* giây .

- **int dppt_putShareFile(char *sendfileName,char *recvfileName,short timeout):**

-Hàm này dùng gởi một file có tên chỉ bởi *sendfileName* đến NSM Server tại vị trí mà maintool thực thi (vị trí mà người sử dụng chạy tool) nếu không thì sẽ gởi file về NSM Server đầu tiên trong danh sách . Tên file nhận được lưu với tên chỉ bởi *recvfileName* . Giá trị *timeout* dùng cho vấn đề timeout .
- Mã trả về :
    - ◊ Thành công :khi initialize thành công thì trả về giá trị TASK_SUCCESSFUL .

◊ Thất bại :nếu quá trình initialization bị lỗi vì một lý do nào đó thì giá trị trả về là TASK_FAIL .

- **int dppt_getShareFile(char *sendfileName,char *recvfileName,short timeout):**

-Hàm này dùng nhận một file có tên chỉ bởi *sendfileName* tại NSM Server ở vị trí mà maintool thực thi (vị trí mà người sử dụng chạy tool) nếu không thì sẽ nhận file ở NSM Server đầu tiên trong danh sách . Tên file nhận về được lưu dưới tên chỉ bởi *recvfileName* . Giá trị *timeout* dùng cho vấn đề timeout .

- Mã trả về :

◊ Thành công :khi initialize thành công thì trả về giá trị TASK_SUCCESSFUL .

◊ Thất bại :nếu quá trình initialization bị lỗi vì một lý do nào đó thì giá trị trả về là TASK_FAIL .

2.2 Các hàm trao đổi message thông qua MsgPassDaemon :

-Trong quá trình trao đổi message thông qua daemon ta định nghĩa một cấu trúc dữ liệu cho data của user như sau :

```
#define HEADERMSG 8
#define MAXMESGDATA 1024
typedef struct  UserInfor {
unsigned int mesg_id :
unsigned int mesg_len:
char mesg_buf[MAXMESGDATA -HEADERMSG]:
}:
```

- Khi sử dụng cấu trúc trên thì khi truyền message đi tool sẽ tự động chuyển đổi các giá trị *mesg_id* và *mesg_len* của message cho phù hợp với biểu diễn data của từng máy .Riêng dữ liệu trong *mesg_buf* thì user phải tự biến đổi cho thích hợp .

với các hàm sau đây :

- **int msgpass_msgget(char *nodename) :**

- Hàm này dùng để user task báo cho hệ thống biết nó sẽ trao đổi message thông qua MsgPass Daemon . Pointer *nodename* chỉ đến chuỗi dùng để xác định cho user task đó . Chuỗi xác định này không được quá 20 ký tự .

- **int msgpass_msgdestroy():**

- Hàm này dùng để báo cho hệ thống biết rằng user task sẽ không tham gia vào quá trình trao đổi message nữa và hệ thống có thể loại bỏ các biến dùng để truy xuất message qua MsgPass Daemon .

- **int msgpass_msgsnd(UserInfor *infor ,char *nodename ,short timeout):**

- Hàm này dùng truyền một message có dạng cấu trúc UserInfor tới user task có tên là *nodename* trong thời gian *timeout* giây .Trong trường hợp *nodename* bằng NULL thì hàm chỉ gởi message mà không quan tâm đến task nhận .Nếu thời gian timeout bằng 0 thì dùng thời gian timeout của hệ thống .

- **int msgpass_msgrcv(UserInfor *infor ,char *nodename ,short timeout):**

- Hàm này dùng nhận một message có dạng cấu trúc UserInfor từ user task có tên là *nodename* trong thời gian *timeout* giây .Trong trường hợp *nodename* bằng NULL thì hàm nhận message bất kỳ ở đầu queue không quan tâm đến task gởi .Nếu thời gian timeout bằng 0 thì dùng thời gian timeout của hệ thống .

2.3 Các hàm trao đổi message trực tiếp :

- Các hàm dưới đây dùng để trao đổi dữ liệu trực tiếp giữa hai user task . Quá trình tương tự như lập trình với socket interface .

- **int msgpass_bind(char *my_nodename):**

-Hàm này dùng để báo cho tool biết user task sẽ trao đổi messsage trược tiếp . Giá trị của pointer my_nodename chỉ tên của task . Hàm này sẽ tạo xác nhận với hệ thống user task này bind với một socket còn trống .

-Mã trả về :

- ◊ Thành công : trả về giá trị fd chỉ đến socket connection dùng trao đổi data .
- ◊ Thất bại : trả về giá trị -10 .

- **int msgpass_accept(int main_msgpass_connect):**

- Hàm này nhận fd chỉ đến socket connection của task và chờ đợi yêu cầu connect của các task khác cần trao đổi qua connection này .Khi có yêu cầu connect tới thì nó sẽ chấp nhận và trả về một cầu nối mới để trao đổi data .

-Mã trả về :

- ◊ Thành công :trả về giá trị chỉ đến fd của socket connection được chấp nhận .
- ◊ Thất bại : trả về giá trị -10 .

- **void msgpass_close(int handleMsgPass):**

- Hàm này dùng đóng một cầu nối được tạo ra bởi msgpass_bind() hoặc msgpass_accept() .Giải phóng cầu nối cho phép tài nguyên của hệ thống được giải phóng .

- **int msgpass_connect(char *des_nodename,short timeout):**

- Hàm này dùng để user task connect tới một user task .Khi connect thành công thì user task sẽ nhận được một cầu nối dùng trao đổi data .Chuỗi *des_nodename* chỉ đến task đích mà sẽ trao đổi . Giá trị *timeout* chỉ thời gian timeout ,nếu timeout bằng 0 thì dùng timeout mặc định của hệ thống .

-Mã trả về :

- ◊ Thành công :trả về giá trị lớn chỉ đến fd của socket mà chỉ connection .
- ◊ Thất bại : trả về giá trị -10 .

- **int msgpass_recv(int msgpass_connect ,char *data ,unsigned int len ,short timeout):**

- Hàm này nhận một cầu nối chỉ bởi giá trị *msgpass_connect* và nhận dữ liệu thông qua cầu nối đó .Khối dữ liệu nhận được có kích thước *len* chứa vào vị trí chỉ bởi pointer *data* . Cầu nối trên được tạo bởi các hàm *msgpass_connect()* hoặc *msgpass_accept()* . Giá trị *timeout* dùng chỉ đến thời gian timeout . Nếu timeout bằng 0 thì hệ thống sử dụng thời gian timeout mặt định .

-Mã trả về :

- ◊ Thành công : trả về số byte nhận được .
- ◊ Thất bại : trả về mã lỗi nhận được .

- **int msgpass_send (int msgpass_connect ,char *data ,unsigned int len ,short timeout):**

- Hàm này nhận một cầu nối chỉ bởi giá trị *msgpass_connect* và gởi dữ liệu thông qua cầu nối đó .Khối dữ liệu gởi được có kích thước *len* ,dữ liệu được chỉ bởi pointer *data* . Cầu nối trên được tạo bởi các hàm *msgpass_connect()* hoặc *msgpass_accept()* . Giá trị *timeout* dùng chỉ đến thời gian timeout . Nếu timeout bằng 0 thì hệ thống sử dụng thời gian timeout mặt định .

-Mã trả về :

- ◊ Thành công : trả về số byte nhận được .
- ◊ Thất bại : trả về mã lỗi nhận được .

2.4 Hàm chuyển đổi dữ liệu sang dạng chuẩn :

- Dưới đây là các hàm chuyển đổi data của tool . Mặc định của tool là không chuyển đổi dữ liệu .Các hàm này tương tự với nhau .
- Mã trả về của các hàm này giống nhau :
  - ◊ Thành công : mã trả về TRUE .
  - ◊ Thất bại : mã trả về là FALSE

2.4.1 Hàm chuyển đổi đơn giản :

- **bool_t Bool_XDR (bool_t *bp_in, bool_t *bp_out):**

- Hàm này chuyển đổi một số bool_t trong ngôn ngữ C sang dạng XDR và ngược lại . Pointer bp_in dùng chỉ số bool_t cần chuyển và pointer bp_out chỉ đến kết quả .

- **bool_t uShort_XDR(unsigned short *sp_in,unsigned short *sp_out):**

- Hàm này chuyển đổi một số unsigned short trong ngôn ngữ C sang dạng XDR và ngược lại . Pointer sp_in dùng chỉ số unsigned short cần chuyển và pointer sp_out chỉ đến kết quả .

- **bool_t Double_XDR(double *dp_in ,double *dp_out):**

- Hàm này chuyển đổi một số double trong ngôn ngữ C sang dạng XDR và ngược lại . Pointer dp_in dùng chỉ số double cần chuyển và pointer dp_out chỉ đến kết quả .

- **bool_t Float_XDR(float *dp_in ,float *dp_out):**

- Hàm này chuyển đổi một số float trong ngôn ngữ C sang dạng XDR và ngược lại . Pointer dp_in dùng chỉ số float cần chuyển và pointer dp_out chỉ đến kết quả .

- **bool_t uInt_XDR(unsigned int *ip_in ,unsigned int *ip_out) :**

- Hàm này chuyển đổi một số unsigned int trong ngôn ngữ C sang dạng XDR và ngược lại . Pointer ip_in dùng chỉ số unsigned int cần chuyển và pointer ip_out chỉ đến kết quả .

- **bool_t Int_XDR(int *ip_in ,int *ip_out):**

- Hàm này chuyển đổi một số int trong ngôn ngữ C sang dạng XDR và ngược lại . Pointer ip_in dùng chỉ số int cần chuyển và pointer ip_out chỉ đến kết quả .

- **bool_t Hyper_XDR(longlong_t *llp_in ,longlong_t *llp_out):**

-Hàm này dùng chuyển một số longlong_t trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer llp_in chỉ đến số cần chuyển , pointer llp_out chỉ kết quả .

- **bool_t uHyper_XDR(u_longlong_t *ulp_in ,u_longlong_t *ulp_out):**

-Hàm này dùng chuyển một dãy các số u_longlong_t trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer ulp_in chỉ đến dãy cần chuyển , pointer ulp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t Long_XDR(long *lp_in ,long *lp_out):**

- Hàm này chuyển đổi một số long trong ngôn ngữ C sang dạng XDR và ngược lại . Pointer lp_in dùng chỉ số long cần chuyển và pointer lp_out chỉ đến kết quả .

- **bool_t uLong_XDR(unsigned long *lp_in ,unsigned long *lp_out):**

- Hàm này chuyển đổi một số unsigned long trong ngôn ngữ C sang dạng XDR và ngược lại . Pointer lp_in dùng chỉ số unsigned long cần chuyển và pointer lp_out chỉ đến kết quả .

- **bool_t uLongLong_XDR (u_longlong_t *ulp_in ,u_longlong_t *ulp_out):**

-Hàm này dùng chuyển một số u_longlong_t trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer ulp_in chỉ đến số cần chuyển , pointer ulp_out chỉ kết quả .

- **bool_t Quadruple_XDR(long double *ldp_in ,long double *ldp_out):**

- Hàm này chuyển đổi một số long double trong ngôn ngữ C sang dạng XDR và ngược lại . Pointer ldp_in dùng chỉ số long double cần chuyển và pointer ldp_out chỉ đến kết quả .

2.4.2 Hàm chuyển đổi dữ liệu theo dãy :

- **bool_t BoolArray_XDR(bool_t *bp_in ,bool_t *bp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số bool_t trong ngôn ngữ CC sang dạng XDR và ngược lại. Pointer bp_in chỉ đến dãy cần chuyển ,pointer bp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t DoubleArray_XDR(double *dp_in ,double *dp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số double trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer dp_in chỉ đến dãy cần chuyển ,pointer dp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t FloatArray_XDR(float *fp_in ,float *fp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số float trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer fp_in chỉ đến dãy cần chuyển , pointer fp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t IntArray_XDR(int *ip_in ,int *ip_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số int trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer ip_in chỉ đến dãy cần chuyển,pointer ip_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t LongArray_XDR(long *lp_in ,long *lp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số long trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer lp_in chỉ đến dãy cần chuyển , pointer lp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t QuadrupleArray_XDR(long double *qp_in ,long double *qp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số long double trong ngôn ngữ CC sang dạng XDR và ngược lại. Pointer qp_in chỉ đến dãy cần chuyển ,pointer qp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t uIntArray_XDR(unsigned int *uip_in ,unsigned int *uip_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số unsigned int trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer uip_in chỉ đến dãy cần chuyển , pointer uip_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t uLongArray_XDR(unsigned long *ulp_in ,unsigned long *ulp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số unsigned long trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer ulp_in chỉ đến dãy cần chuyển , pointer ulp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t uShortArray_XDR (unsigned short *usp_in ,unsigned short *usp_out , unsigned int n) :**

-Hàm này dùng chuyển một dãy các số unsigned short trong ngôn ngữ CC sang dạng XDR và ngược lại. Pointer usp_in chỉ đến dãy cần chuyển , pointer usp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t HyperArray_XDR(longlong_t *hp_in ,longlong_t *hp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số longlong_t trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer hp_in chỉ đến dãy cần chuyển , pointer hp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t LongLongArray_XDR(longlong_t *llp_in ,longlong_t *llp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số longlong_t trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer llp_in chỉ đến dãy cần chuyển , pointer llp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t uHyperArray_XDR(u_longlong_t *uhp_in ,u_longlong_t *uhp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số bool_t trong ngôn ngữ CC sang dạng XDR và ngược lại. Pointer bp_in chỉ đến dãy cần chuyển ,pointer bp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

- **bool_t uLongLongArray_XDR (u_longlong_t * ullp_in , u_longlong_t * ullp_out ,unsigned int n) :**

-Hàm này dùng chuyển một dãy các số u_longlong_t trong ngôn ngữ C sang dạng XDR và ngược lại. Pointer ullp_in chỉ đến dãy cần chuyển , pointer ullp_out chỉ kết quả và n là số phần tử của dãy cần chuyển .

PHẦN 5

TỔNG KẾT CÔNG VIỆC VÀ HƯỚNG PHÁT TRIỂN CỦA ĐỀ TÀI

1. Các công việc đã hoàn tất

DPPT Version 1.0 đã thể hiện được nhiều đặc điểm nổi bật của một một phần mềm hỗ trợ cho xử lý song song trên hệ thống phân bố và đã đem đến cho các user một công cụ khá mạnh , tiện lợi khi họ muốn giải quyết một vấn đề lớn trên hệ thống mạng phân bố Unix .

Sau đây là các công việc mà DPPT Version 1.0 đã hoàn tất :

- *Về phần giao diện đồ họa với người sử dụng (GUI) :*

  - Cho phép user mô hình hóa các module trong bản thiết kế ứng dụng của mình thành các node trong đồ thị ứng dụng có cấu trúc dạng cây được thể hiện dưới dạng đồ họa (vùng Application) .
  - Cho phép user thay đổi việc mô hình hóa nói trên với các chức năng như delete node , delete sub-graph , chuyển node hay một sub-graph từ một node tới một node khác trong đồ thị ứng dụng , ...
  - Mỗi node có một bảng lưu các tính chất của chúng như : tên file của chương trình nguồn cần soạn thảo (hoặc đã có sẵn) , trình biên dịch cho chương trình nguồn của node này , các thư viện khi biên dịch , ...
  - User có thể soạn thảo source code cho một node hay lấy lại các file đã soạn thảo để gán cho node đó qua khai báo trong properties của node .
  - Cho phép một node có thể chỉ là một hàm chứ không cần là một chương trình hoàn chỉnh và sau đó DPPT sẽ tự động sinh code để tạo ra một chương trình hoàn chỉnh cho node (chức năng generate) .
  - Cho phép user có thể map các node của ứng dụng lên các host khác nhau trên mạng .
  - Cung cấp chức năng compile các node của ứng dụng tại các host mà node được map lên tạo ra khả năng cho DPPT có thể chạy được trên một hệ thống phân bố không đồng nhất (homogeneous) .
  - Cung cấp chức năng thực thi toàn bộ ứng dụng hoặc từng node của ứng dụng .

- User có thể lưu lại công việc đang làm và phục hồi lại trong phiên làm việc sau với chức năng save ,save as và open .
- Có chức năng *map tự động* các module lên các host trong hệ thống .
- Có chức năng cho phép user detect lại các host trong hệ thống và tìm ra được các host đã chết so với lần detect ngay trước đó .
- ...

Về phần hệ thống :

- Lấy và hiển thị được cấu hình hệ thống mạng Unix : DPPT tìm kiếm và hiển thị cho user thấy các host trong hệ thống với các màu sắc khác nhau thể hiện được trạng thái của các host .
- DPPT software có đặc tính multiuser : đã giải quyết được vấn đề tranh chấp ,xung đột hệ thống khi có nhiều user cùng chạy DPPT (việc đặt tên file khi biên dịch từ xa , tạo ra các port của DPPT , ...) .
- Hỗ trợ cho cả mô hình Shared-Memory và mô hình Message Passing .
- DPPT có một thư viện cho người lập trình với những hàm chức năng khá mạnh và đa dạng .
- ...

Ngoài ra chúng tôi còn viết ba demo để minh họa cho DPPT Version 1.0 và chương trình nguồn của ba demo này được kèm theo trong đĩa mềm cùng với báo cáo này :

- Một demo minh họa cho các chức năng cơ bản của toàn bộ phần GUI trong DPPT.
- Demo thứ hai là một chương trình xử lý ảnh Mandelbrot , demo này không dùng các hàm thư viện của DPPT mà chỉ liên lạc giữa các node bằng truyền file data và dùng DPPT như một công cụ để phân bổ các module lên các host trên hệ thống được thuận tiện và dễ dàng .
- Demo thứ ba là một chương trình tận dụng hầu như toàn bộ sức mạnh của DPPT để tính tích phân của một hàm số .

Ngoài ra còn có nhiều dự định trong khi thiết kế và thực thi DPPT nhưng do thời gian có hạn nên không thể hoàn thành hoặc giải quyết chưa được tối ưu . Sau đây chúng tôi sẽ đưa ra những vấn đề còn tồn tại với DPPT cũng như những ý tưởng mới trong thiết kế đồng thời cũng đề nghị hướng giải quyết , phát triển nhằm tạo ra một DPPT trong tương lai ngày càng hoàn hảo hơn và mạnh hơn .

2. Hướng phát triển của đề tài

- Công cụ cần hỗ trợ cả các host có nhiều CPU (multiprocessor) : ngày nay công nghệ phát triển mạnh mẽ , giá thành của các máy multiprocessor cũng ngày càng giảm xuống do đó DPPT cũng cần phải được cải tiến để đáp ứng được nhu cầu của người sử dụng trong tương lai .

- Cần có thêm chức năng gỡ rối (debug) cho ứng dụng của user : lập trình xử lý song song trên mạng phân bố là một vấn đề khó khăn , phức tạp , nhất là có lỗi xảy ra khi thực thi chương trình ứng dụng (runtime error) do đó debug là một vấn đề rất cần thiết để đem lại sự hoàn chỉnh cho một DPPT trong tương lai .
- Với cách compile các node của DPPT hiện thời chúng ta thấy còn có một hạn chế là nếu một node được map giữa các host cùng loại CPU (đồng nhất) thì DPPT vẫn tiến hành biên dịch lại module của node đó nên tạo ra sự lãng phí về thời gian dẫn tới làm giảm hiệu xuất làm việc của user trên DPPT . Trong tương lai DPPT nên biên dịch theo chủng loại CPU : mỗi host nên có mục nhận dạng loại CPU của nó để tiện lợi cho việc biên dịch , chỉ cần dịch trên một host và sau đó nếu node được map sang host khác cùng loại CPU (hoặc tương thích) thì chương trình thực thi của node không cần dịch lại mà chỉ cần copy từ host cũ sang host mà node mới được map lên .
- Hiện thời tất cả các node tích cực trong đồ thị ứng dụng của DPPT đều có thể truy xuất tới một biến chung bất kể là nó được khai báo trong node nào của đồ thị ứng dụng . Điều này không hợp lý với mô hình quản lý biến của các ngôn ngữ lập trình thông thường như C , C++ , Pascal , ... tức là các node chỉ được truy xuất tới các biến chung trong phạm vi mà node đó được phép , ví dụ node 2 là con của node 1 , node 4 là con của node 3 và node 1,3 là ngang cấp thì các biến khai báo trong node 1 chỉ được node 2 và các node con cháu khác của node 1 truy xuất chứ node 4 không thể truy xuất được các biến chung loại này . Trong tương lai DPPT nên giải quyết vấn đề này để mô hình của bài toán được tạo ra với DPPT sẽ gần với thực tế hơn cũng như dễ hiểu hơn đối với người lập trình và đồng thời cũng dễ sửa chương trình hơn nếu gặp lỗi.
- Hiện thời thuật giải map tự động các node lên các host của DPPT còn đơn giản , tương lai DPPT cần phải tìm được topology của mạng , đo được hiệu xuất đường truyền , hiệu xuất sử dụng CPU của các máy cũng như thời gian delay trên mạng , ... để từ đó có được thuật giải map tự động các node một cách hợp lý và chính xác hơn nữa nhằm đem lại lời giải tối ưu cho bài toán song song trên mọi phương diện .
- Hiện tại DPPT chỉ giải được một bài toán tại một thời điểm : chỉ có một vùng Application . Trong tương lai DPPT cần cho phép user thực hiện được nhiều bài toán một lúc (có nhiều vùng Application) . Ý tưởng này sẽ đem lại cho các nhà nghiên cứu khi làm việc với DPPT nhiều thuận lợi : có nhiều vấn đề có liên quan đến nhau sẽ được user thử cùng một lúc để bổ sung cho nhau nhằm đưa ra quyết định cuối cùng được tối ưu nhất .
- Cần có nhiều chức năng quản lý và điều khiển các node trong quá trình ứng dụng đang được thực thi (runtime) như : stop một module , stop ứng dụng , chuyển node sang host khác nếu host hiện tại có tải quá nặng , ...
- Tương lai DPPT cũng nên có chức năng phát hiện ra các host chết trong khi ứng dụng thực thi để đưa các node được map lên host đó sang một host khác còn sống : hiện tại DPPT chỉ hiển thị các host chết cho user thấy (nếu có) mỗi khi user detect lại các host trong hệ thống mạng .
- Đối vối mô hình Message Passing : thực hiện việc truyền message theo group .Xây dựng việc truyền nhận message qua MsgPassDaemon nhưng có cơ chế reply .

- Hướng phát triển việc truyền nhận message thông qua các stream và có thể xây dựng một device riêng dùng stream để truyền nhận message .
- Nếu có các công cụ lớp dưới có sẵn cho việc xây dựng DSM hay Message Passing thì sử dụng chúng để xây dựng hệ thống tối ưu hơn .
- Phát triển thư viện cho những ngôn ngữ khác như Fortran , Java ...

TÀI LIỆU THAM KHẢO

1. X Window System Programming and Applications with Xt OSF/MOTIF EDITION
   (Douglas A.Young - 1990)

2. Unix Network Programming
   (W. Richard Stevens - 1990)

3. Distributed Systems Concept and Design
   (George Coulouris , Jean Dollimore and Tim Kindberg - 1994)

4. UNIX Programmer's Reference
   (John Valley - 1991)

5. Unix Programmer's Quick Reference
   (John Valley - 1990)

6. Using Unix
   (Larry Schumer , Chris Negus with Dave Gunter - 1995)

7. Internetworking with TCP/IP (volume I,II,III)
   Design , Implementation and Internals
   (Douglas E. Comer , David L. Stevens - 1994)

8. Solaris Multithreaded Programming Guide
   (SunSoft - 1995)

9. Tạp chí *" IEEE Transactions on Parallel and Distributed Systems , Vol 7 , NO 4 , April 1996 "* .

10. Computer Graphics
    (F.S Hill , JR. - 1990)

11. Cấu trúc dữ liệu
    (Nguyễn Trung Trực - 1993)

12. Multithreading Programming Techniques
    (Shashi Prasad - 1997)

13. Interprocess Communication in UNIX

(John Sharepley Gray - 1997)

14. Unix System V/386 User‘s Guide
(AT&T - 1988)

15. Unix SystemV/386 Programmer’s Reference Manual
(AT&T - 1998)

16. UNIX internal the new frontiers
(Uresh Vahalia - 1996)

17. Modern Openrating Systems
(Andrew S. Tanenbaum - 1992)

18. Computer Achitecture and Parallel Processing
(Kai Hwang & Fayé A. Briggs - 1987)