TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN MÔN HỌC

CƠ SỞ NGÀNH MẠNG

**ĐỀ TÀI 1: BÀI TOÁN NĂM TRIẾT GIA ĂN TỐI**

**ĐỀ TÀI 2: SỬ DỤNG PIPELINE TRONG JAVA XÂY DỰNG CHƯƠNG TRÌNH TRAO ĐỔI THÔNG TIN THEO MÔ HÌNH CLIENT-SERVER**

**Giáo viên hướng dẫn: ThS. TRẦN HỒ THỦY TIÊN**

**Sinh viên thực hiện: LƯU THỊ NGỌC LAN**

**Lớp: 14T2**

**MSSV: 102140076**

Đà Nẵng, 3/2018

**LỜI CẢM ƠN**

Nguyên lý hệ điều hành và Lập trình mạng là những kiến thức căn bản là nền tảng mà mỗi lập trình viên phải hiểu rõ để phục vụ cho những kiến thức cao hơn.

Hệ điều hành là tập hợp các chương trình phần mềm chạy trên máy tính, dùng để điều hành, quản lý các thiết bị phần cứng và tài nguyên phần mềm trên máy tính. Hệ điều hành đóng vai trò trung gian trong việc giao tiếp giữa người sử dụng và phần cứng máy tính, cung cấp một môi trường cho phép người sử dụng phát triển và thực hiện các ứng dụng một cách dễ dàng.

Cùng với sự phát triển của mạng Internet hiện nay, thì việc nghiên cứu nằm vững những kiến thức về mạng là rất quan trọng.

Dựa trên những hiểu biết của mình và tìm hiểu tài liệu, em đã nghiên cứu và thực hiện 2 đề tài: Bài toán 5 triết gia ăn tối và xây dựng chương trình trao đổi thông tin giữa Client- Server sử dụng Pipeline.

Em xin chân thành cảm ơn thầy cô khoa Công Nghệ Thông Tin đã tạo điều kiện để em nghiên cứu kĩ hơn những kiến thức này, và đặc biệt là cô Trần Hồ Thủy Tiên đã nhiệt tình theo dõi, hướng dẫn em trong quá trình thực hiện đề tài này.

Vì kiến thức còn hạn hẹp, nên không thể tránh khỏi những sai sót trong quá trình làm đề tài, rất mong nhận được sự góp ý của thầy cô để sản phầm được hoàn thiện hơn. Em xin chân thành cảm ơn.

Sinh viên thực hiện

Lưu Thị Ngọc Lan

MỤC LỤC

[MỞ ĐẦU ………………………………………………………………………….1](#_Toc515520244)

[PHẦN I: NGUYÊN LÍ HỆ ĐIỀU HÀNH 2](#_Toc515520245)

[CHƯƠNG 1: CƠ SỞ LÍ THUYẾT 3](#_Toc515520246)

[1. Tiến trình (Proccess) 3](#_Toc515520247)

[2. Tài nguyên găng và đoạn găng 4](#_Toc515520248)

[3. Giải pháp Semaphone 6](#_Toc515520249)

[4. Deadlock 6](#_Toc515520250)

[CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG 8](#_Toc515520251)

[1. Cách giải quyết bài toán 8](#_Toc515520252)

[2. Sơ đồ bài toán 8](#_Toc515520253)

[CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 9](#_Toc515520254)

[1. Môi trường 9](#_Toc515520255)

[2. Triển khai 9](#_Toc515520256)

[KẾT QUẢ ĐẠT ĐƯỢC 11](#_Toc515520257)

[1. Vấn đề còn tồn tại 11](#_Toc515520258)

[2. Hướng phát triển 11](#_Toc515520259)

[PHẦN II: LẬP TRÌNH MẠNG 12](#_Toc515520260)

[CHƯƠNG 1: CƠ SỞ LÍ THUYẾT 13](#_Toc515520261)

[1. Kĩ Thuật Ống Dẫn Pipe Line Trong Java 13](#_Toc515520262)

[2. Mô Hình Ứng Dụng Client-Server 14](#_Toc515520263)

[CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG 17](#_Toc515520264)

[1. Bài toán: 17](#_Toc515520265)

[2. Sơ đồ hoạt động 17](#_Toc515520266)

[CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 18](#_Toc515520267)

[1. Môi trường 18](#_Toc515520268)

[2. Triển khai 18](#_Toc515520269)

[KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 20](#_Toc515520270)

[1. Kết Quả Đạt Được 20](#_Toc515520271)

[2. Hướng Phát Triển 20](#_Toc515520272)

[TÀI LIỆU THAM KHẢO 21](#_Toc515520273)

[PHỤ LỤC ………………………………………………………………………...22](#_Toc515520274)

**DANH SÁCH HÌNH ẢNH**

*Hình 1: Minh họa bài toán 5 triết gia ăn tối 2*

*Hình 2. Tiến trình song song trong hệ thống uniprocessor 5*

*Hình 3. Tiến trình song song trong hệ thống multiprocessor 5*

*Hình 4. Semaphores 8*

*Hình 5. Cấu trúc một chương trình trong giải pháp semaphore 9*

*Hình 6. Cấu trúc chương trình trong giải pháp semaphore 9*

*Hình 7. Đồ thị cấp phát tài nguyên 11*

*Hình 8. Sơ đồ hoạt động của mỗi triết gia 14*

*Hình 9: Màn hình khởi tạo bài toán 16*

*Hình 10. Các tiến trình đang chạy 16*

*Hình 11: Cách tạo ống dẫn trong java 19*

*Hình 12: Mô hình Client-Server 21*

*Hình 13: Chế độ giao tiếp nghẽn 22*

*Hình 14: Chế độ giao tiếp không nghẽn 23*

*Hình 15: Sơ đồ hoạt động của ứng dụng client-server 25*

*Hình 16: Giao diện chat client 27*

*Hình 17: Giao diện chat client 27*

*Hình 18: Nội dung chat 28*

DANH SÁCH TỪ VIẾT TẮT

|  |  |
| --- | --- |
| **Từ viết tắt** | **Diễn giải** |
| JDT | Java Development Toolkit |
| SDK | Software Development Kit |
| PDE | Plug-in Development Environment |

MỞ ĐẦU

1. **Tổng quan về đề tài**

**2. Mục đích và ý nghĩa của đề tài**

***2.1. Mục đích***

- Đi sâu và nẵm vững một cách có hệ thống kiến thức đã thu nhận được trong quá trình học lí thuyết, làm bài tập và thực hành.

- Từng bước làm quen với các công tác khoa học có định hướng của giáo viên hướng dẫn và hình thành hành vi nghiên cứu độc lập có sự trợ giúp của tài liệu tham khảo.

- Gắn quá trình học lí thuyết với công tác nghiên cứu thực tế.

- Trình bày rõ ràng và khoa học một vấn đề thuộc lĩnh vực nghiện cứu của mình.

***2.2. Ý nghĩa***

- Kết quả của công việc phản ánh công sức, tài năng, trí tuệ của người làm đồ án và phải được trình bày bằng văn bản trong đồ án môn học theo những chuẩn mực và yêu cầu của giáo viên hướng dẫn.

- Đồ án môn học phản ánh công sức nghiên cứu nên cần phải được trình bày trước bộ môn và được đánh giá bằng điểm số. Việc trình bày đúng, rõ ràng, ngắn gọn và khoa học chỉ có được ở những sinh viên có quá trình lao động nghiêm túc cho đề tài và có chuẩn bị đầy đủ kiến thức trong lĩnh vực nghiên cứu đề tài.

**3. Bố cục của đề tài**

Báo cáo đề tài bao gồm các nội dung sau:

*Mở đầu*

*Phần I: Nguyên lý hệ điều hành*

*Phần II: Lập trình mạng*

*Kết luận chung*

*Tài liệu tham khảo*

*Phụ lục*

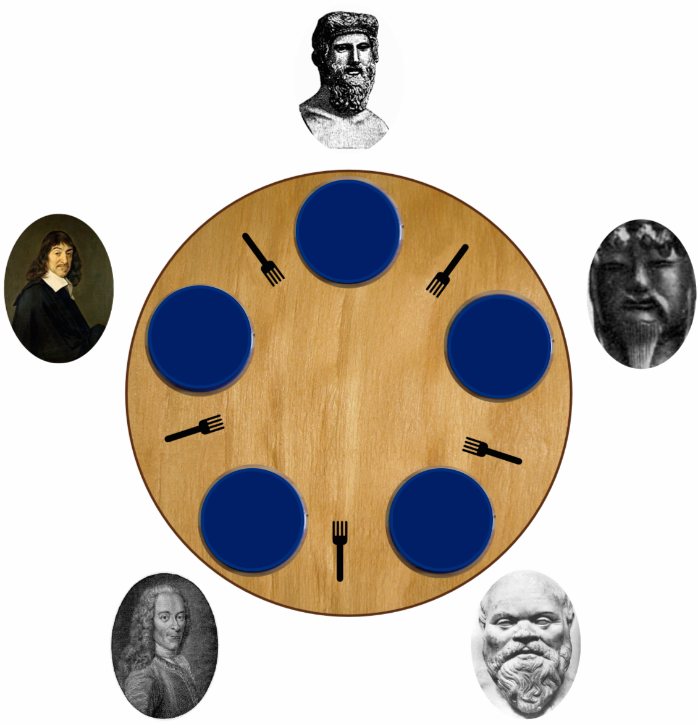
# 

# PHẦN I: NGUYÊN LÍ HỆ ĐIỀU HÀNH

**Đề tài:** Viết chương trình giải quyết bài toán năm triết gia ăn tối. Chương trình phải tạo ra năm tiến trình con mô phỏng hoạt động của năm triết gia. Dùng semaphore để đồng bộ hoạt động của năm triết gia này.

**Mô tả vấn đề:**

Đây là bài toán cổ điển về hệ điều hành. Bài toán bữa tối của cá triết gia được đưa ra bởi nhà toán học E. W. Dijkstra. Bài toán được mô tả như sau:

Có năm triết gia cùng ngồi ăn tối quanh một chiếc bàn tròn, trước mặt mỗi người có một đĩa mì Ý, giữa 2 triết gia có một chiếc nĩa.

*Hình 1: Minh họa bài toán 5 triết gia ăn tối*

Mỗi triết gia dành toàn bộ thời gian để suy nghĩ hoặc ăn khi đói.

Mối triết gia chỉ có thể ăn khi có được 2 chiếc nĩa bên cạnh mình.

Đói: một triết gia có thể chết đói nếu ông ta không có cách nào để ăn được.

Tắc nghẽn: các triết gia phải đợi lẫn nhau nên không có ai ăn được.

**Yêu cầu bài toán:**

Phải đặt ra thuật toán sao cho khi một triết gia đói thì ông ta sẽ được ăn và đảm bảo không có triết gia nào bị chết đói.

Bài toán đặt ra vấn đề “đồng bộ giữa các tiến trình”, giải quyết vấn đề tắc nghẽn có thể xảy ra.

Thuật toán được đưa ra là thuật toán Semaphore.

## CHƯƠNG 1: CƠ SỞ LÍ THUYẾT

### Tiến trình (Proccess)

* 1. **Khái niệm:**

Tiến trình là một bộ phận của một chương trình đang thực hiện, đơn vị thực hiện tiến trình là Proccesser.

Vì tiến trình là một bộ phận của chương trình nên tương tự như chương trình, tiến trình cũng sở hữu một con trỏ lệnh, một con trỏ Stack, một tập các thanh ghi, một không gian địa chỉ trong bộ nhớ chính và tất cả các thông tin cần thiết khác để tiến trình có thể hoạt động được.

* 1. **Định nghĩa tiến trình**

Theo Saltzer: Tiến trình là một chương trình do một proccesor logic thực hiện.

Theo Horming& Rendell: Tiến trình là một quá trình chuyển từ trạng thái này sang trạng thái khác dưới tác động của hàm hành động, xuất phát từ một trạng thái ban đầu nào đó.

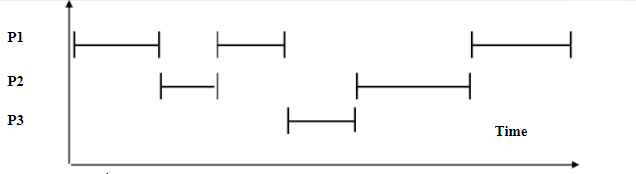
* 1. **Các loại tiến trình**

Tiến trình tuần tự: là các tiến trình mà điểm khởi tạo nó là điểm kết thúc của tiến trình trước đó.

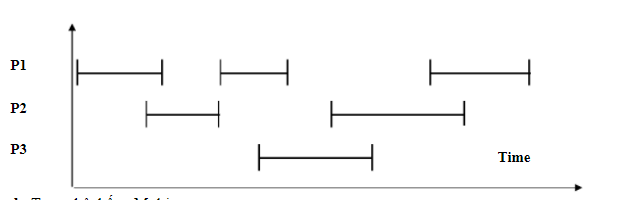
Tiến trình song song: là các tiến trình mà điểm khởi tạo của tiến trình này nằm ở thân của tiến trình khác, tức là có thể khởi tạo một tiến trình mới khi các tiến trình trước đó chưa kết thúc.

Tiến trình tuần tự xuất hiện trong các hệ điều hành đơn nhiệm như hệ điều hành MS\_DOS.

Các tiền trình song song xuất hiện trong hệ điều hành đa nhiệm (uniprocessor và multiprocesser).



Hình 2. Tiến trình song song trong hệ thống uniprocessor



Hình 3. Tiến trình song song trong hệ thống multiprocessor

* 1. **Tuyến (Thread)**

Tuyến là một thành phần của tiến trình sở hữu ngăn xếp và thực thi độc lập ngay trong mã lệnh của tiến trình. Nếu như hệ điều hành có nhiều tiến trình thì trong mỗi tiến trình hoạt động song song trong hệ điều hành. Ưu điểm của tuyến là chúng hoạt động trong cùng một không gian địa chỉ của tiến trình. Tập hợp một nhóm các tuyến có thể sử dụng chung biến toàn cục, vùng nhớ heap, bằng mô tả file, … của tiến trình, cơ chế liên lạc giữa các tuyến đơn giản hơn cơ chế liên lạc giữa các tiến trình với nhau (nếu hệ điều hành của bạn chạy trên phần cứng nhiều bộ xử lí thì tuyến thực sự chạy song song chứ không phải chạy giả lập kiều xoay vòng).

Ưu điểm: sử dụng tuyến trong tiến trình đơn giản hơn lập trình tuần tự. Nhiều thao tác xuất nhập hoặc hiển thị dữ liệu có thế tách rời và phân cho các tuyến chạy được độc lập thực thi. Ví dụ trong môi trường đồ họa, khi bạn copy một file có kích thước lớn, chương trình sẽ được thiết kế sao cho một tuyến thực hiện đọc ghi dữ liệu từ đĩa, tuyến khác sẽ đảm trách việc hiển thị phần trăm hoàn thành công việc cho người dùng theo dõi tiến độ.

Đối với hệ điều hành chi phí để chuyển đổi giữa ngữ cảnh của tiến trình cao và chậm hơn chi phí chuyển đổi ngữ cảnh đảm bảo cho tuyến (với tiến trình hệ điều hành phải cất thông số môi trường, thanh ghi trạng thái, hoán đổi vùng nhớ, …).

Tuy nhiên, điểm yếu của việc dùng tuyến đó là khả năng đổ vỡ của một tuyến sẽ ảnh hướng đến tất cả các tuyến khác và toàn bộ tiến trình đang hoạt động. Lí do là các tuyến dùng chung vùng nhớ và không gian địa chỉ của tiến trình. Ngược lại, một tiến trình bị đổ vỡ luôn được hệ điều hành cô lập hoàn toàn không gây ảnh hưởng đến các tiến trình khác. Tiến trình có thể chạy trên nhiều máy khác nhau trong khi tuyến chỉ được thực thi trên một máy và trong một tiến trình.

### Tài nguyên găng và đoạn găng

**2.1. Tài nguyên găng (Critical Resource)**

Trong môi trường hệ điều hành đa nhiệm- đa chương- đa người dùng sử dụng, việc chia sẻ tài nguyên cho các tiến trình của người dùng sử dụng dùng chung là cần thiết, nhưng nếu hệ điều hành không tổ chức tốt việc sử dụng tài nguyên dùng chung của các tiến trình hoạt động đồng thời, thì không những không mang lại hiệu quả khai thác tài nguyên của hệ thống mà còn làm hỏng dữ liệu của các ứng dụng. Và nguy hiểm hơn là việc hỏng dữ liệu này có thể hệ điều hành và ứng dụng không phát hiện được. Việc hỏng dữ liệu của ứng dụng có thể làm sai lệch ý nghĩa thiết kế của nó. Đây là điều mà cả hệ điều hành và người lập trình đều không mong muốn.

Các tiến trình hoạt động đồng thời thường cạnh tranh với nhau trong việc sử dụng tài nguyên dùng chung. Hai tiến trình hoạt động đồng thời cùng ghi vào một không gian nhớ chung (một biến chung) trên bộ nhớ hay hai tiến trình đồng thời cùng ghi dữ kiệu vào một file chia sẻ, đó là những biểu hiện của sự cạnh tranh về việc sử dụng tài nguyên dùng chung của các tiến trình. Để các tiến trình hoạt động đồng thời không cạnh tranh hay xung đột với nhau khi sử dụng tài nguyên dùng chung hệ điều hành phải tổ chức cho các tiến trình này được độc quyên truy xuất/ sử dụng trên các tài nguyên dùng chung này.

Những tài nguyên được hệ điều hành chia sẻ cho nhiều tiến trình hoạt động đồng thời dùng chung, mà có nguy co dẫn đến sự tranh chấp giữa các tiến trình này khi sử dụng chúng, được gọi là tàu nguyên găng. Tài nguyên găng có thể là tài nguyên phần cứng hoặc tài nguyên phần mềm, có thể là tài nguyên phân chia được hoặc không phân chia được, nhưng đa số thường là tài nguyên phân chia được như là: các biến chung, các file chia sẻ.

**2.2. Đoạn găng (Critical Section)**

Đoạn code trong các tiến trình đồng thời, có tác động đến các tài nguyên có thể trở thành tài nguyên găng được gọi là đoạn găng hay miền găng. Tức là, các đoạn code trong các chường trình dùng để truy cập đến các vùng nhớ chia sẻ, các tập tin chia sẻ được gọi là các đoạn găng.

Để hạn chế các lỗi có thể xảy ra do sử dụng tài nguyên găng, hệ điều hành phải điều khiển các tiến trình sao cho, tại một thời điểm chỉ có một tiến trình nằm trong đoạn găng, nếu có nhiều tiến trình cùng muốn vào( thực hiện) đoạn găng thì chỉ có một tiến trình được vào, các tiến trình khác phải chờ, một tiến trình khi ra khỏi( kết thúc) đoạn găng phải báo cho hệ điều hành và/ hoặc các tiến trình khác biết để các tiến trình này vào đoạn găng,vv. Các công tác điều khiến tiến trình được thực hiện đoạn găng của hệ điều hành được gọi là điều độ tiến trình qua đoạn găng. Để công tác điều độ tiến trình qua đoạn găng được thành công, thì cần phải có sự phối hợp giữa vi xử lí, hệ điều hành và người lập trình. Vi xử lí đưa các chỉ thị, hệ điều hành cung cấp các công cụ để người lập trình cây dựng các sơ đồ điều độ hợp lý, để đảm bảo sự độc quyền trong việc sử dụng tài nguyên găng của tiến trình.

**2.3. Yêu cầu đối với đoạn găng**

Đoạn găng phải thỏa mãn các yêu cầu sau:

- Tại một thời điểm không thể có hai tiến trình nằm trong đoạn găng.

- Nếu có nhiều tiến trình đồng thời cùng xin được vào đoạn găng thì chỉ có một tiến trình được phép vào đoạn găng, các tiến trình khác phải xếp hàng chờ trong hàng đợi.

- Tiến trình chờ ngoài đoạn găng không được ngăn cản các tiến trình khác vào đoạn găng.

- Không có tiến trình nào được phép ở lâu vô hạn trong đoạn găng và không có tiến trình phải chờ lâu mới được vào đoạn găng (chờ trong hàng đợi).

- Nếu tài nguyên găng được giải phóng thì hệ điều hành có nhiệm vụ đánh thức các tiến trình trong hàng đợi ra để tạo điều kiện cho nó vào đoạn găng.

Các vấn đề có thể gặp phải đối với đoạn găng:

* Có thể dẫn đến tắc nghẽn(Deadlock) trong hệ thống.
* Các tiến trình có thể bị đói(Stravation) tài nguyên.

### Giải pháp Semaphone

**3.1.  Tiếp cận**: Được Dijkstra đề xuất vào 1965, một semaphore s là một biến có các thuộc tính sau:

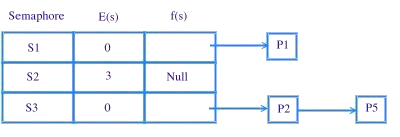
-       Một giá trị nguyên dương e(s)

-       Một hàng đợi f(s) lưu danh sách các tiến trình đang bị khóa (chờ) trên semaphore s

-       Chỉ có hai thao tác được định nghĩa trên semaphore:

o   Down(s): giảm giá trị của semaphore s đi 1 đơn vị nếu semaphore có trị e(s) > 0, và tiếp tục xử lý. Ngược lại, nếu e(s) ≤ 0, tiến trình phải chờ đến khi e(s) >0.

o   Up(s): tăng giá trị của semaphore s lên 1 đơn vị. Nếu có một hoặc nhiều tiến trình đang chờ trên semaphore s, bị khóa bởi thao tácDown, thì hệ thống sẽ chọn một trong các tiến trình này để kết thúc thao tác Down và cho tiếp tục xử lý.

[](http://2.bp.blogspot.com/-PFPRrteT61I/VIBQMcGFYMI/AAAAAAAABQU/eXTYMzRVvG8/s1600/310+Semaphores.jpg)

*Hình 4.  Semaphores*

|  |  |
| --- | --- |
| Down(s):  e(s) = e(s) - 1;  if e(s) < 0 {  status(P)= blocked;  enter(P,f(s));  } | Up(s):  e(s) = e(s) + 1;  if s ≤ 0 {  exit(Q,f(s)); //Q là tiến trình đang chờ trên s  status (Q) = ready;  enter(Q,ready-list);  } |

**3.2. Cài đặt:** Gọi p là tiến trình thực hiện thao tác Down(s) hay Up(s).

Lưu ý cài đặt này có thể đưa đến một giá trị âm cho semaphore, khi đó trị tuyệt đối của semaphore cho biết số tiến trình đang chờ trên semaphore.

Điều quan trọng là các thao tác này cần thực hiện một cách không bị phân chia, không bị ngắt nửa chừng, có nghĩa là không một tiến trình nào được phép truy xuất đến semaphore nếu tiến trình đang thao tác trên semaphore này chưa kết thúc xử lý hay chuyển sang trạng thái blocked.

**3.3.  Sử dụng:** có thể dùng semaphore để giải quyết vấn đề truy xuất độc quyền hay tổ chức phối hợp giữa các tiến trình.

**3.3.1. Tổ chức truy xuất độc quyền với Semaphores:** khái niệm semaphore cho phép bảo đảm nhiều tiến trình cùng truy xuất đến miền găng mà không có sự mâu thuẫn truy xuất. N tiến trình cùng sử dụng một semaphore s, e(s) được khởi gán là 1. Để thực hiện đồng bộ hóa, tất cả các tiến trình cần phải áp dụng cùng cấu trúc chương trình sau đây:

|  |
| --- |
| while (TRUE)   {  Down(s)  critical-section ();  Up(s)  Noncritical-section ();  } |

*Hình 5.  Cấu trúc một chương trình trong giải pháp semaphore*

**3.3.2. Tổ chức đồng bộ hóa với Semaphores:** với semaphore có thể đồng bộ hóa hoạt động của hai tiến trình trong tình huống một tiến trình phải đợi một tiến trình khác hoàn tất thao tác nào đó mới có thể bắt đầu hay tiếp tục xử lý. Hai tiến trình chia sẻ một semaphore s, khởi gán e(s) là 0. Cả hai tiến trình có cấu trúc như sau:

|  |  |
| --- | --- |
| P1:  while (TRUE) {  job1(); Up(s); //đánh thức P2  } | P2:  while (TRUE) {  Down(s); // chờ P1 job2();  } |

*Hình 6.  Cấu trúc chương trình trong giải pháp semaphore*

**3.3.3. Thảo luận:** Nhờ có thực hiện một cách không thể phân chia, semaphore đã giải quyết được vấn đề tín hiệu "đánh thức" bị thất lạc. Tuy nhiên, nếu lập trình viên vô tình đặt các primitive Down và Up sai vị trí, thứ tự trong chương trình, thì tiến trình có thể bị khóa vĩnh viễn.

Ví dụ: *while (TRUE) {*

*Down(s)  
critical-section ();  
Noncritical-section ();*

*}*

tiến trình trên đây quên gọi Up(s), và kết quả là khi ra khỏi miền găng nó sẽ không cho tiến trình khác vào miền găng!

Vì thế việc sử dụng đúng cách semaphore để đồng bộ hóa phụ thuộc hoàn toàn vào lập trình viên và đòi hỏi lập trình viên phải hết sức thận trọng.

### Deadlock

**4.1. giới thiệu vấn đề**

Trong môi trường đa chương, nhiều quá trình có thể cạnh tranh một số giới hạn tài nguyên. Một quá trình yêu cầu tài nguyên, nếu tài nguyên không sẵn dùng tại thời điểm đó, quá trình đi vào trạng thái chờ. Qúa trình chờ có thể không bao giờ chuyển trạng thái trở lại vì tài nguyên chúng yêu cầu bị giữ bởi những quá trình đang chờ khác. Trường hợp này được gọi là deadlock (khóa chết).

**4.2. Điều kiện hình thành tắt nghẽn**

Năm 1971, Coffiman đã đưa ra và chứng tỏ được rằng nếu hệ thống tồn tại đồng thời bốn điều kiện sau đây thì hệ thống sẽ xảy ra tắt nghẽn:

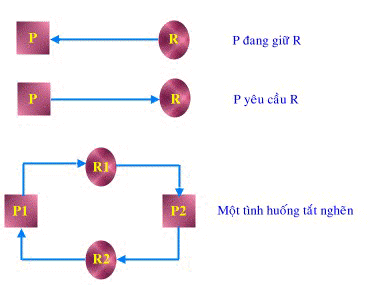
* Loại trừ lẫn nhau (mutual excution) hay độc quyền sử dụng: Đối với các tài nguyên không phân chia được thì tại mỗi thời điểm chỉ có một tiến trình sử dụng được tài nguyên.
* Giữ và đợi (hold and wait): một tiến trình hiện tại đang chiếm giữ tài nguyên, lại xin cấp phát thêm tài nguyên mới.
* Không ưu tiên (no preemption): Không có tài nguyên nào có thể được giải phóng từ một tiến trình đang chiếm giữ nó.

Sự tắc nghẽn có thể tồn tại với ba điều kiện trên, nhưng cũng có thể không xảy ra chỉ với 3 điều kiện đó. Để chắc chắn tắc nghẽn xảy ra cần phải có điều kiện thứ tư:

* Đợi vòng tròn (Circular wait): Đây là trường hợp của ví dụ 1 mà chúng ta đã nêu ở trên. Tức là mỗi tiến trình đang chiếm giữ tài nguyên mà tiến trình khác đang cần.

**4.3. Đồ thị cấp phát tài nguyên**

Có thể sử dụng một đồ thị để mô hình hóa việc cấp phát tài nguyên. Đồ thị này có 2 loại nút: các tiến trình được biểu diễn bằng hình tròn, và mỗi tài nguyên được hiển thị bằng hình vuông.

[](http://1.bp.blogspot.com/-t9evqKaxPpU/VIBQokVpZgI/AAAAAAAABQk/gn54S-povKs/s1600/319+Do+thi+cap+phat+tai+nguyen.png)

*Hình 7.  Đồ thị cấp phát tài nguyên*

**4.4. Ngăn chặn tắc nghẽn (Deadlock Prevention)**

Ngăn chặn tắc nghẽn là thiết kế một hệ thống sao cho hiện tượng tắc nghẽn bị loại trừ. Các phương thức ngăn chặn đều tập trung giải quyết bốn điều kiện gây ra tắc nghẽn, sao cho hệ thống không thể xảy ra đồng thời bốn điều kiện tắc nghẽn:

Tài nguyên không thể chia sẻ: nhìn chung gần như không thể tránh được điều kiện này vì bản chất tài nguyên gần như cố định. Tuy nhiên đối với một số tài nguyên về kết xuất, người ta có thể dùng các cơ chế spooling để biến đổi thành tài nguyên có thể chia sẻ.

 Sự chiếm giữ và yêu cầu thêm tài nguyên: phải bảo đảm rằng mỗi khi tiến trình yêu cầu thêm một tài nguyên thì nó không chiếm giữ các tài nguyên khác. Có thể áp đặt một trong hai cơ chế truy xuất sau:

Tiến trình phải yêu cầu tất cả các tài nguyên cần thiết trước khi bắt đầu xử lý.

* Phương pháp này có khó khăn là tiến trình khó có thể ước lượng chính xác tài nguyên cần sử dụng vì có thể nhu cầu phụ thuộc vào quá trình xử lý. Ngoài ra nếu tiến trình chiếm giữ sẵn các tài nguyên chưa cần sử dụng ngay thì việc sử dụng tài nguyên sẽ kém hiệu quả.
* Khi tiến trình yêu cầu một tài nguyên mới và bị từ chối, nó phải giải phóng các tài nguyên đang chiếm giữ, sau đó lại được cấp phát trở lại cùng lần với tài nguyên mới.

 - Phương pháp này làm phát sinh các khó khăn trong việc bảo vệ tính toàn vẹn dữ liệu của hệ thống.

Không thu hồi tài nguyên: Cho phép hệ thống được thu hồi tài nguyên từ các tiến trình bị khóa và cấp phát trở lại cho tiến trình khi nó thoát khỏi tình trạng bị khóa. Tuy nhiên với một số loại tài nguyên, việc thu hồi sẽ rất khó khăn vì vi phạm sự toàn vẹn dữ liệu.

  Tồn tại một chu kỳ: Tránh tạo chu kỳ trong đồ thị bằng cách cấp phát tài nguyên theo một sự phân cấp như sau:

Gọi R = {R1, R2, ..., Rm} là tập các loại tài nguyên.

Các loại tài nguyên được phân cấp từ 1-N.

Ví dụ: *F(đĩa) = 2, F (máy in) = 12*

Các tiến trình khi yêu cầu tài nguyên phải tuân thủ quy định: khi tiến trình đang chiếm giữ tài nguyên Ri thì chỉ có thể yêu cầu các tài nguyên Rj nếu F(Rj) > F(Ri).

## CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

### Cách giải quyết bài toán

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

* 1. **Quản lí vùng găng**

Chương trình xem mối triết gia là một tiến trình, chopstick là tài nguyên chung cần được bảo vệ.

Biết mutex được xây dựng trong class Chopstisk để quản lí tài nguyên dùng chung là các chopstisk, mỗi chopstisk được tạo sẽ có 1 biến khóa Semaphores mutex để đánh dấu:

* mutex.availablePermits()>0: chopstisk đang ở trạng thái free.
* mutex.availablePermits()<0: chopstisk đang được sử dụng

Class Chopstisk đóng vai trò quản lí vùng găng với 2 phương thức synchronized acquire() và release() để đóng và mở vùng găng.

* 1. **Ngăn chặn tắc nghẽn**
* Mỗi Philosopher sẽ được khởi tạo với 2 chopstick:

**public** Philosopher(**int** id, Chopstick leftChopstick, ChopstickrightChopstick) {

**this**.id = id;

**this**.leftChopstick = leftChopstick;

**this**.rightChopstick = rightChopstick;

}

* Hành vi của mỗi triết gia được mô tả như sau:

**public** **void** run() {

**try** {

**while** (**true**) {

think();

pickUpChopstick(leftChopstick,rightChopstick);

eat();

putDownChopsticks();

}

} **catch** (InterruptedException e) {

}

}

### Sơ đồ bài toán

F

While(true)

Think()

PickUpChopstick()

Check available chopstick

Eat()

putDownChopstick()

T

T

*Hình 8.  Sơ đồ hoạt động của mỗi triết gia*

**CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ**

### Môi trường

* 1. **Eclipse IDE**

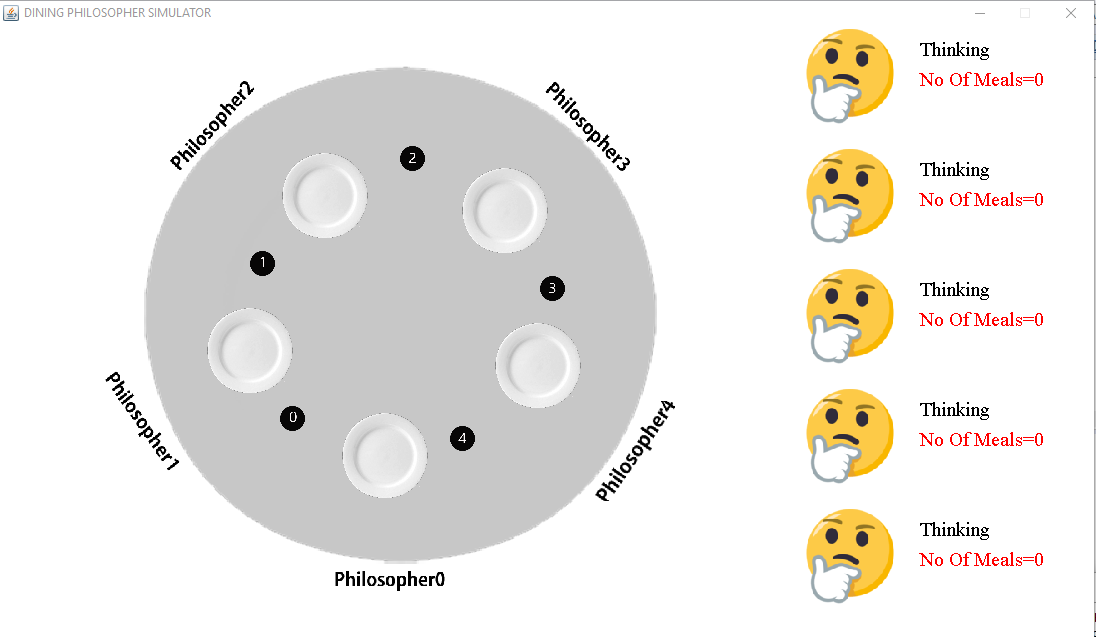
Eclipse là phần mềm miễn phí, được các nhà phát triển sử dụng để xây dựng những ứng dụng J2EE, sử dụng Eclipse nhà phát triển có thể tích hợp với nhiều công cụ hỗ trợ khác để có được một bộ công cụ hòan chỉnh mà không cần dùng đến phần mềm riêng nào khác. Eclipse SDK bao gồm 3 phần chính: Platform, Java Development Toolkit (JDT), Plug-in Development Environment (PDE). Với JDT, Eclipse được xem như là một môi trường hỗ trợ phát triển Java mạnh mẽ. PDE hỗ trợ việc mở rộng Eclipse, tích hợp các Plug-in vào Eclipse Platform. Eclipse Platform là nền tảng của toàn bộ phần mềm Eclipse, mục đích của nó là cung cấp những dịch vụ cần thiết cho việc tích hợp những bộ công cụ phát triển phần mếm khách dưới dạng Plug-in, bản thân JDT cũng có thể được coi như là một Plug-in làm cho Eclipse như là một Java IDE (Integrated Development Enviroment).

* 1. **JRE**

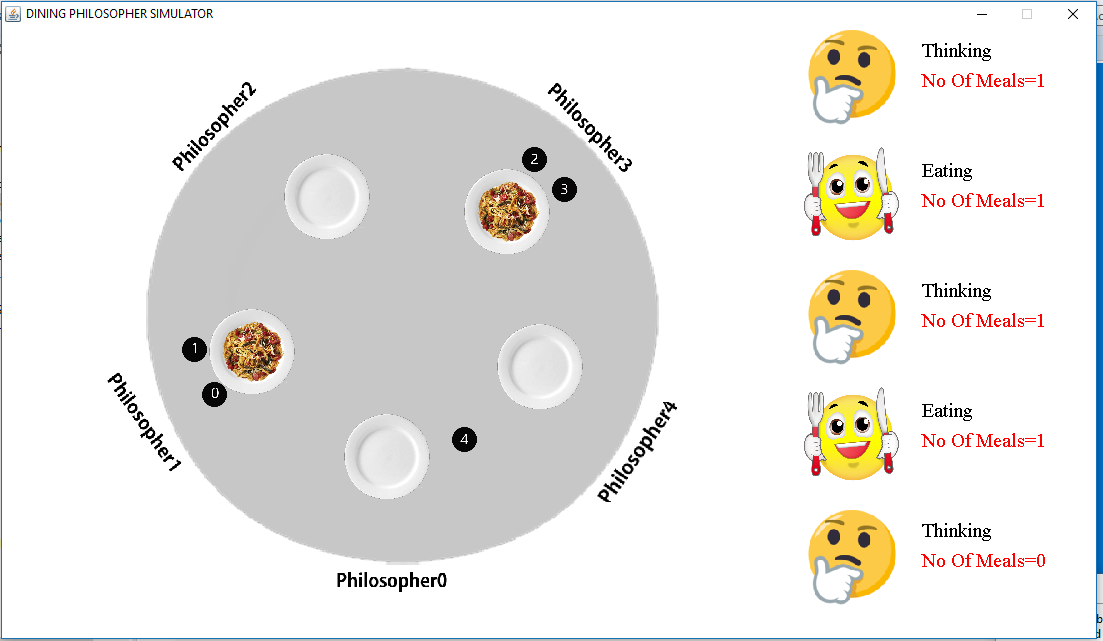
JRE (là viết tắt của Java Runtime Environment) được sử dụng để cung cấp môi trường runtime. Nó là trình triển khai của JVM. JRE bao gồm tập hợp các thư viện và các file khác mà JVM sử dụng tại runtime. Trình triển khai của JVM cũng được công bố bởi các công ty khác ngoài Sun Micro Systems.

### Triển khai

* Class Chopstick.java: tài nguyên dùng chung.
* Class Philosopher.java: class tạo các tiến trình.
* Class Display.java: class xây dựng giao diện.
* Class App.java: class tạo đối tượng chạy chương trình.

1. **Kết quả**

*Hình 9: Màn hình khởi tạo bài toán*



*Hình 10. Các tiến trình đang chạy*

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### Kết Quả Đạt Được

Thông qua đồ án này em đã hiểu được kĩ hơn về giải pháp semaphore cũng như việc tranh chấp tài nguyên dùng chung giữa các tiến trình và cách giải quyết những tranh chấp đó.

Bên cạnh những mặt đã đạt được ở trên, trong quá trình làm đồ án chúng em đã học thêm được nhiều kiến thức bổ ích về hệ điều hành cũng như việc áp dụng giải pháp semaphore phát triển dựa trên ngôn ngữ lập trình Java.

### Vấn đề còn tồn tại

* Giao diện và hình ảnh còn đơn giản.

### Hướng phát triển

* Tiếp tục tìm hiểu nghiên cứu sâu hơn về Thread và các giải pháp tránh tắc nghẽn.
* Chỉnh sửa giao diện cho thuận tiện và logic hơn.

# PHẦN II: LẬP TRÌNH MẠNG

**Đề tài:** Sử dụng Pipe line trong Java xây dựng chương trình trao đổi thông tin theo mô hình Cient-Server.

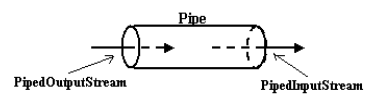
Nội dung đề tài:

* Server:
* Chờ nhận các byte gửi đến từ client.
* Khi có byte đến, nhận byte và gửi về client chính byte đó.
* Client:
* Gửi từng kí tự đến server và chờ nhận kết quả gửi về.
* Chương trình minh họa:
* Sử dụng pipe làm phương tiện giao tiếp giữa Client và Server.
* Server là một thread thực thi song song với Client
* Client nhận các kí tự từ bàn phím và gửi cho server, hiển thị kết quả nhận được ra màn hình.
* Server đổi kí tự nhận được thành kí tự HOA trước khi gửi về cho client.

## CHƯƠNG 1: CƠ SỞ LÍ THUYẾT

### Kĩ Thuật Ống Dẫn Pipe Line Trong Java

* 1. **Giới thiệu**
* Java hỗ trợ tiên ích ống dẫn thông qua hai lớp java.io.PipedInputStream và java.io.PipedOutputStream. Chúng là hai đầu của một ống dẫn. Trong đó PipedInputStream là đầu đọc dữ liệu và PipedOutputStream là đầu ghi dữ liệu của ống dẫn.
* PipedInputStream là lớp con của InputStream nên nó có tất cả các thuộc tính của InputStream.
* PipedOutputStream là lớp con của OutputStream nên nó có tất cả các thuộc tính của OutputStream.
  1. **Các cách tạo ống dẫn**
* Để tạo một ống dẫn ta chỉ cần tạo ra hai đối tượng thuộc lớp PipedInputStream và PipedOutputStream và nối chúng lại với nhau. Khi đó dữ liệu được ghi vào PipedOutputStream sẽ được đọc ra ở đầu PipedInputStream:



*Hình 11: Cách tạo ống dẫn trong java*

* **Cách 1:**

Bước 1: Tạo đầu đọc:

PipedInputStream readId = new PipedInputStream();

Bước 2: Tạo đầu ghi:

PipedOutputStream writeId = new PipedOutputStream();

Bước 3: Nối đầu đọc với đầu ghi hay ngược lại

readId.connect(writeId);

// hoặc writeId.connect(readId);

* **Cách 2**

Bước 1: Tạo đầu đọc:

PipedInputStream readId = new PipedInputStream();

Bước 2: Tạo đầu ghi và nối vào đầu đọc đã có:

PipedOutputStream writeId = new PipedOutputStream(readId);

Hoặc:  Ta có thể tạo đầu ghi trước rồi tạo đầu đọc sau.

* Lưu ý: Các phương thức khởi tạo của PipedInputStream và PipedOutputStream sử dụng ở trên đòi hỏi phải "bắt" (catch) IOException do chúng có thể “quẳng” ra (throws).

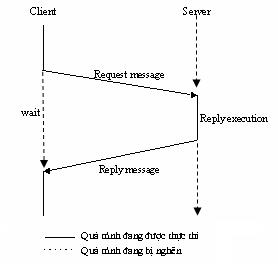
### Mô Hình Ứng Dụng Client-Server

**2.1. Thành phần:**

- Quá trình chuyên cung cấp một số phục vụ nào đó, chẳng hạn: phục vụ tập tin, phục vụ máy in, phục vụ thư điện tử, phục vụ Web... Các quá trình này được gọi là các trình phục vụ hay Server.

* Một số quá trình khác có yêu cầu sử dụng các dịch vụ do các server cung cấp được gọi là các quá trình khách hàng hay Client.

Việc giao tiếp giữa client và server được thực hiện dưới hình thức trao đổi các thông điệp (Message). Để được phục vụ, client sẽ gởi một thông điệp yêu cầu (Request Message) mô tả về công việc muốn server thực hiện. Khi nhận được thông điệp yêu cầu, server tiến hành phân tích để xác định công việc cần phải thực thi. Nếu việc thực hiện yêu cầu này có sinh ra kết quả trả về, server sẽ gởi nó cho client trong một thông điệp trả lời (Reply Message). Dạng thức (format) và ý nghĩa của các thông điệp trao đổi giữa client và server được qui định rõ bởi giao thức (protocol) của ứng dụng.



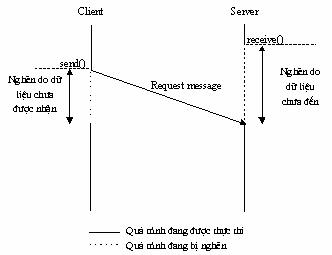
*Hình 12: Mô hình Client-Server*

**2.2. Các chế độ giao tiếp**

Quá trình giao tiếp giữa client và server có thể diễn ra theo hai chế độ là nghẽn (blocked) hay không nghẽn (Non blocked).

* Chế độ nghẽn:

Trong chế độ này, khi quá trình client hay server phát ra lệnh gởi dữ liệu, (thông thường bằng lệnh send), sự thực thi của nó sẽ bị tạm dừng cho đến khi quá trình nhận phát ra lệnh nhận số dữ liệu đó (thường là lệnh receive).

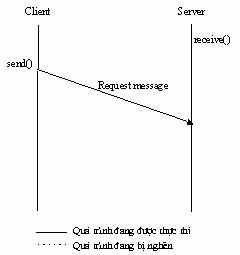


*Hình 13: Chế độ giao tiếp nghẽn*

Tương tự cho trường hợp nhận dữ liệu, nếu  quá trình nào đó, client hay server, phát ra lệnh nhận dữ liệu, mà ở thời điểm đó chưa có dữ liệu gởi đến, sự thực thi của nó cũng tạm dừng cho đến khi có dữ liệu gởi đến.

* Chế độ không nghẽn:

Trong chế độ này, khi quá trình client hay server phát ra lệnh gởi dữ liệu, sự thực thi của nó vẫn được tiếp tục mà không quan tâm đến việc có quá trình nào phát ra lệnh nhận số dữ liệu đó hay không.



*Hình 14: Chế độ giao tiếp không nghẽn*

Tượng tự cho trường hợp nhận dữ liệu, khi quá trình phát ra lệnh nhận dữ liệu, nó sẽ nhận được số lượng dữ liệu hiện có (bằng 0 nếu chưa có quá trình nào gởi dữ liệu đến). Sự thực thi của quá trình vẫn được tiếp tục.

Trong thực tế cần chú ý đến chế độ giao tiếp nghẽn khi lập trình, vì nó có thể dẫn đến trường hợp chương trình bị "treo" do số lần gởi và nhận giữ liệu không bằng nhau giữa hai bên giao tiếp.

## CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

### Bài toán:

Sử dụng Pipe line trong Java xây dựng chương trình trao đổi thông tin theo mô hình Cient-Server.

1. **Thiết kế:**

Sử dụng kĩ thuật pipeline trong java:

* Tạo 2 tuyến con độc lập, tuyến Client và tuyến Server, đầu đọc của Client sẽ nối với đầu ghi của Server và ngược lại, đầu ghi của Client sẽ nối với đầu đọc của Server- sử dụng phương thức readId.connect(writeId);
* Override phương thức run() để lấy dữ liệu từ đầu đọc (read) khi có dữ liệu gửi từ đầu ghi (write)

@Override

**public** **void** run() {

**while**(**true**){

readMessage();

}

}

**private** **void** readMessage() {

**byte**[] buf = **new** **byte**[1024];

**try** {

**int** len = in.read(buf);

System.***out***.println("Client: " + **new** String(buf, 0, 100));

receiveDataFromClient(**new** String(buf, 0, 100));

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

}

}

* Tạo action khi nhấn nút send, tiến hành ghi dữ liệu vào đầu ghi của đường ống gửi sang đầu đọc – sử dụng lệnh out.write(Byte[] data);

@Override

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getActionCommand()=="Send"){

**try** {

writeMessage(txtMessage.getText());

sendMessageToClient();

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

}

**private** **void** writeMessage(String tem) {

**try** {

out.write(tem.getBytes());

} **catch** (IOException e) {

e.printStackTrace();

}

}

### Sơ đồ hoạt động

**in**

**out**

Client

Server

**readMessage()**

**writeMessage(msg)**

**send()**

toUpperCaseMsg()

**writeMessage(msg)**

**send()**

**Connect()**

**Connect()**

**in**

**out**

**readMessage()**

**readMessage()**

*Hình 15: Sơ đồ hoạt động của ứng dụng client-server*

## CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

### Môi trường

* 1. **Eclipse IDE**

Eclipse là phần mềm miễn phí, được các nhà phát triển sử dụng để xây dựng những ứng dụng J2EE, sử dụng Eclipse nhà phát triển có thể tích hợp với nhiều công cụ hỗ trợ khác để có được một bộ công cụ hòan chỉnh mà không cần dùng đến phần mềm riêng nào khác. Eclipse SDK bao gồm 3 phần chính: Platform, Java Development Toolkit (JDT), Plug-in Development Environment (PDE). Với JDT, Eclipse được xem như là một môi trường hỗ trợ phát triển Java mạnh mẽ. PDE hỗ trợ việc mở rộng Eclipse, tích hợp các Plug-in vào Eclipse Platform. Eclipse Platform là nền tảng của toàn bộ phần mềm Eclipse, mục đích của nó là cung cấp những dịch vụ cần thiết cho việc tích hợp những bộ công cụ phát triển phần mếm khách dưới dạng Plug-in, bản thân JDT cũng có thể được coi như là một Plug-in làm cho Eclipse như là một Java IDE (Integrated Development Enviroment).

* 1. **JRE**

JRE (là viết tắt của Java Runtime Environment) được sử dụng để cung cấp môi trường runtime. Nó là trình triển khai của JVM. JRE bao gồm tập hợp các thư viện và các file khác mà JVM sử dụng tại runtime. Trình triển khai của JVM cũng được công bố bởi các công ty khác ngoài Sun Micro Systems.

### Triển khai

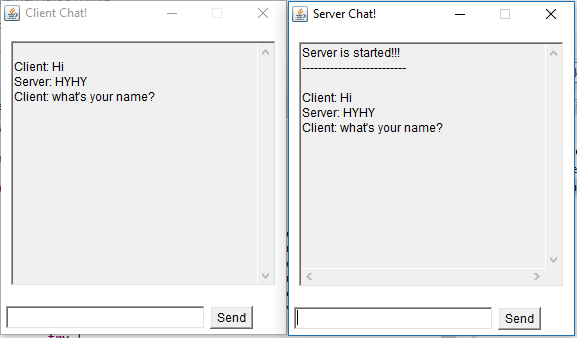
* Class ClientUI.java: chứa giao diện chat client và các phương thức gửi, nhận dữ liệu
* Class ServerUI.java: chứa giao diện chat Server và các phương thức gửi, nhận dữ liệu từ server.
* Class PipedTest.java: tạo 2 tuyến client và server chạy song song, kết nối đầu đọc với đầu ghi.

1. **Kết quả**

*Hình 16: Giao diện chat client*



*Hình 17: Giao diện chat client*



*Hình 18: Nội dung chat*

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### Kết Quả Đạt Được

Trong thời gian tìm hiểu, nghiên cứu cơ sở lý thuyết và triển khai ứng dụng công nghệ, đề tài đã đạt được những kết quả sau:

*Về mặt lý thuyết,* hiểu được cách thức hoạt động của Pipeline, client-server và đa luồng.

*Về mặt thực tiễn ứng dụng*, ứng dụng vào việc giao tiếp giữa client-server

Tuy nhiên, đề tài còn tồn tại các vấn đề như sau:

* Vấn đề 1: Pipeline thường không được ưu tiên sử dụng trong trao đổi client-server vì nó không kết nối các máy khác IP.
* Vấn đề 2: Giao diện còn đơn giản

### Hướng Phát Triển

Một số số hướng nghiên cứu và phát triển của đề tài như sau:

* Bổ sung các chức năng trao đổi dữ liệu trong trao đổi client-server.

TÀI LIỆU THAM KHẢO

**Tiếng Việt**

[1] Đặng Vũ Tùng, Giáo trình nguyên lý hệ điều hành, NXB Hà Nôi,  
2005.  
[2] Trần Hồ Thuỷ Tiên, Bài giảng Nguyên lý hệ điều hành, Khoa CNTT  
trường Đại học Bách khoa Đà nẵng.

**Internet**

1. [*http://codebyshaukat.blogspot.com/2014/11/dining-philosophers-problem.html*](http://codebyshaukat.blogspot.com/2014/11/dining-philosophers-problem.html)
2. [*https://gist.github.com/Alexey-N-Chernyshov/16198f75b284191bf2406d27eaad6b23*](https://gist.github.com/Alexey-N-Chernyshov/16198f75b284191bf2406d27eaad6b23)
3. [*http://blog.vnamct.com/2014/04/he-dieu-hanh-chuong-5-cac-gia-phap-ong.html*](http://blog.vnamct.com/2014/04/he-dieu-hanh-chuong-5-cac-gia-phap-ong.html)

PHỤ LỤC

**1. Dining Philosophy**

**- Chopstick.java**

import java.util.concurrent.Semaphore;

public class Chopstick {

private int id;

private static int *xpos*[] = new int[5];

private static int *ypos*[] = new int[5];

public Semaphore mutex = new Semaphore(1);

void grab() {

try {

mutex.acquire();

} catch (Exception e) {

e.printStackTrace(System.*out*);

}

}

void release() {

mutex.release();

}

boolean isFree() {

return mutex.availablePermits() > 0;

}

public Chopstick() {

}

public Chopstick(int id, int xpos, int ypos) {

this.id = id;

this.*xpos*[id] = xpos;

this.*ypos*[id] = ypos;

}

public static void setAxis(int id, int x, int y) {

*xpos*[id] = x;

*ypos*[id] = y;

}

public int getXpos(int id) {

return *xpos*[id];

}

public int getYpos(int id) {

return *ypos*[id];

}

}

**- Philosopher.java**

**import** java.util.Random;

**public** **class** Philosopher **implements** Runnable {

**private** **int** id;

**final** **static** **int** ***THINKING*** = 0;

**final** **static** **int** ***EATING*** = 3;

**final** **static** **int** ***LEFT\_BUSY*** = 2;

**final** **static** **int** ***RIGHT\_BUSY*** = 1;

**private** Chopstick leftChopstick;

**private** Chopstick rightChopstick;

**private** **static** **int** *noMeals*[] = **new** **int**[5];

**private** Random timeGenerator = **new** Random();

**private** **static** **int** *state*[] = **new** **int**[5];

**private** **static** **int** *quote*[] = **new** **int**[5];

**private** Chopstick chopstick = **new** Chopstick();

**public** Philosopher() {

}

**public** Philosopher(**int** id, Chopstick leftChopstick, Chopstick rightChopstick) {

**this**.id = id;

**this**.leftChopstick = leftChopstick;

**this**.rightChopstick = rightChopstick;

}

**public** **void** run() {

**try** {

**while** (**true**) {

think();

pickUpChopstick(leftChopstick,rightChopstick);

eat();

putDownChopsticks();

}

} **catch** (InterruptedException e) {

System.***out***.println("Philosopher " + id + " was interrupted.\n");

}

}

**private** **void** think() **throws** InterruptedException {

System.***out***.println("Philosopher " + id + " is thinking.\n");

System.***out***.flush();

**if** (id == 0) {

*state*[0] = 4;

}

**if** (id == 1) {

*state*[1] = 4;

}

**if** (id == 2) {

*state*[2] = 4;

}

**if** (id == 3) {

*state*[3] = 4;

}

**if** (id == 4) {

*state*[4] = 4;

} //

Thread.*sleep*(2000 + timeGenerator.nextInt(1000));

}

**private** **void** pickUpChopstick(Chopstick leftChopstick, Chopstick rightChopstick) {

**while** (**true**) {

**try** {

leftChopstick.grab();

rightChopstick.grab();

} **finally** {

**if** (!leftChopstick.isFree() && !rightChopstick.isFree()) {

System.***out***.println("Philosopher " + id + " Pick Up both Chop Stick.\n");

*quote*[id] = 3;

**return**;

}

**if** (!leftChopstick.isFree()) {

System.***out***.println("Philosopher " + id + " Pick Up leftChop Stick And RightChop Stick is Busy.\n");

leftChopstick.release();

System.***out***.println("Philosopher " + id + " Put Down leftChop Stick.\n");

*quote*[id] = 1;

}

**if** (!rightChopstick.isFree()) {

System.***out***.println("Philosopher " + id + " Pick Up RightChop Stick And lefttChop Stick is Busy.\n");

rightChopstick.release();

System.***out***.println("Philosopher " + id + " Put Down RightChop Stick.\n");

*quote*[id] = 2;

}

}

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**private** **void** eat() **throws** InterruptedException {

Thread.*sleep*(500);

*quote*[id] = 0;

increment(id);

System.***out***.println("Philosopher " + id + " is eating.\n");

**if** (id == 0) {

chopstick.*setAxis*(0, 340, 470);

chopstick.*setAxis*(4, 383, 475);

*state*[0] = 0;

}

**if** (id == 1) {

chopstick.*setAxis*(0, 200, 355);

chopstick.*setAxis*(1, 180, 310);

*state*[1] = 0;

}

**if** (id == 2) {

chopstick.*setAxis*(1, 250, 140);

chopstick.*setAxis*(2, 280, 110);

*state*[2] = 0;

}

**if** (id == 3) {

chopstick.*setAxis*(2, 520, 120);

chopstick.*setAxis*(3, 550, 150);

*state*[3] = 0;

}

**if** (id == 4) {

chopstick.*setAxis*(3, 585, 340);

chopstick.*setAxis*(4, 560, 375);

*state*[4] = 0;

}

System.***out***.flush();

Thread.*sleep*(400);

//

**if** (id == 0) {

*state*[0] = 1;

}

**if** (id == 1) {

*state*[1] = 1;

}

**if** (id == 2) {

*state*[2] = 1;

}

**if** (id == 3) {

*state*[3] = 1;

}

**if** (id == 4) {

*state*[4] = 1;

}

//

Thread.*sleep*(400);

//

**if** (id == 0) {

*state*[0] = 2;

}

**if** (id == 1) {

*state*[1] = 2;

}

**if** (id == 2) {

*state*[2] = 2;

}

**if** (id == 3) {

*state*[3] = 2;

}

**if** (id == 4) {

*state*[4] = 2;

} //

//

Thread.*sleep*(400);

//

**if** (id == 0) {

*state*[0] = 3;

}

**if** (id == 1) {

*state*[1] = 3;

}

**if** (id == 2) {

*state*[2] = 3;

}

**if** (id == 3) {

*state*[3] = 3;

}

**if** (id == 4) {

*state*[4] = 3;

} //

Thread.*sleep*(400 + timeGenerator.nextInt(200));

}

**private** **void** putDownChopsticks() {

leftChopstick.release();

rightChopstick.release();

System.***out***.println("Philosopher " + id + " put down Both ChopStick\n");

**if** (id == 0) {

chopstick.*setAxis*(0,280, 380);

chopstick.*setAxis*(4, 450, 400);

*state*[0] = 4;

}

**if** (id == 1) {

chopstick.*setAxis*(0, 280, 380);

chopstick.*setAxis*(1, 250, 225);

*state*[1] = 4;

}

**if** (id == 2) {

chopstick.*setAxis*(1, 250, 225);

chopstick.*setAxis*(2, 400, 120);

*state*[2] = 4;

}

**if** (id == 3) {

chopstick.*setAxis*(2, 400, 120);

chopstick.*setAxis*(3, 540, 250);

*state*[3] = 4;

}

**if** (id == 4) {

chopstick.*setAxis*(3, 540, 250);

chopstick.*setAxis*(4, 450, 400);

*state*[4] = 4;

}

}

**public** **int** getS(**int** id) {

**return** *state*[id];

}

**public** **void** setS(**int** id, **int** s) {

*state*[id] = s;

}

**public** **void** increment(**int** id) {

*noMeals*[id]++;

}

**public** **int** getNoMeals(**int** id) {

**return** *noMeals*[id];

}

**public** **int** getQuote(**int** id) {

**return** *quote*[id];

}

}

**- Display.java**

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.Graphics;

**import** java.awt.Image;

**import** java.awt.Toolkit;

**import** javax.swing.JButton;

**import** javax.swing.JPanel;

**public** **class** Display **extends** JPanel **implements** Runnable {

Image bg;

Image s0,s1,s2,s3,s4;

Image name0, name1, name2, name3, name4;

Image plate[]=**new** Image[5];

Toolkit t;

Image state1,state2;

Font font;

Chopstick chopstick;

Philosopher philosopher;

**public** Display(){

t=Toolkit.*getDefaultToolkit*();

font = **new** Font("Serif", Font.***PLAIN***, 20);

bg=t.getImage("image//BG1.png");

s0=t.getImage("image//ts0.png");

s1=t.getImage("image//ts1.png");

s2=t.getImage("image//ts2.png");

s3=t.getImage("image//ts3.png");

s4=t.getImage("image//ts4.png");

name0=t.getImage("image//name0.png");

name1=t.getImage("image//name1.png");

name2=t.getImage("image//name2.png");

name3=t.getImage("image//name3.png");

name4=t.getImage("image//name4.png");

state1=t.getImage("image//eating.jpg");

state2=t.getImage("image//thinking.jpg");

plate[0]=t.getImage("image//p1.png");

plate[1]=t.getImage("image//p2.png");

plate[2]=t.getImage("image//p3.png");

plate[3]=t.getImage("image//p4.png");

plate[4]=t.getImage("image//p5.png");

chopstick=**new** Chopstick();

philosopher=**new** Philosopher();

}

**public** **void** run(){

**while**(**true**){

repaint();

**try** {

Thread.*sleep*(10);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

**public** **void** paint(Graphics g){

**super**.paint(g);

**this**.setBackground(Color.***white***);

g.drawImage(bg,100 , 0, 600, 580, **null**);

drawP(g);

drawName(g);

drawP0(g);

drawP1(g);

drawP2(g);

drawP3(g);

drawP4(g);

draw0(g);

draw1(g);

draw2(g);

draw3(g);

draw4(g);

drawS0(g);

drawS1(g);

drawS2(g);

drawS3(g);

drawS4(g);

}

**public** **void** draw0(Graphics g){

g.drawImage(s0,chopstick.getXpos(0), chopstick.getYpos(0), 25, 25, **null**);

}

**public** **void** draw1(Graphics g){

g.drawImage(s1,chopstick.getXpos(1) , chopstick.getYpos(1), 25, 25, **null**);

}**public** **void** draw2(Graphics g){

g.drawImage(s2,chopstick.getXpos(2), chopstick.getYpos(2), 25, 25, **null**);

}**public** **void** draw3(Graphics g){

g.drawImage(s3,chopstick.getXpos(3), chopstick.getYpos(3), 25, 25, **null**);

}**public** **void** draw4(Graphics g){

g.drawImage(s4,chopstick.getXpos(4), chopstick.getYpos(4), 25, 25, **null**);

}

**public** **void** drawP0(Graphics g){

g.drawImage(plate[philosopher.getS(0)], 335,380, 100, 100, **null**);

}

**public** **void** drawP1(Graphics g){

g.drawImage(plate[philosopher.getS(1)], 200,275, 100, 100, **null**);

}

**public** **void** drawP2(Graphics g){

g.drawImage(plate[philosopher.getS(2)], 275,120, 100, 100, **null**);

}

**public** **void** drawP3(Graphics g){

g.drawImage(plate[philosopher.getS(3)], 455,135, 100, 100, **null**);

}

**public** **void** drawP4(Graphics g){

g.drawImage(plate[philosopher.getS(4)], 488,290, 100, 100, **null**);

}

**public** **void** drawP(Graphics g){

g.drawImage(plate[0], 335,380, 100, 100, **null**);

g.drawImage(plate[0], 200,275, 100, 100, **null**);

g.drawImage(plate[0], 275,120, 100, 100, **null**);

g.drawImage(plate[0], 455,135, 100, 100, **null**);

g.drawImage(plate[0], 488,290, 100, 100, **null**);

}

**public** **void** drawName(Graphics g){

g.drawImage(name0, 335,545, 110, 20, **null**);

g.drawImage(name1, 105,345, 75, 100, **null**);

g.drawImage(name2, 170,55, 85, 90, **null**);

g.drawImage(name3, 545,55, 85, 90, **null**);

g.drawImage(name4, 590,375, 90, 100, **null**);

}

//

**public** **void** drawS0(Graphics g){

g.setFont(font);

//getNoMeals

g.setColor(Color.***RED***);

g.drawString("No Of Meals="+Integer.*toString*(philosopher.getNoMeals(0)), 920, 60);

g.setColor(Color.***BLACK***);

**if**(philosopher.getS(0)==4){

g.drawImage(state2,800 , 0, 100, 100, **null**);

g.drawString("Thinking", 920, 30);

}**else**{

g.drawImage(state1,800 , 0, 100, 100, **null**);

g.drawString("Eating", 920, 30);

}

}

**public** **void** drawS1(Graphics g){

g.setFont(font);

g.setColor(Color.***RED***);

g.drawString("No Of Meals="+Integer.*toString*(philosopher.getNoMeals(1)), 920, 180);

g.setColor(Color.***BLACK***);

**if**(philosopher.getS(1)==4){

g.drawImage(state2,800 , 120, 100, 100, **null**);

g.drawString("Thinking", 920, 150);

}**else**{

g.drawImage(state1,800 , 120, 100, 100, **null**);

g.drawString("Eating", 920, 150);

}

}

**public** **void** drawS2(Graphics g){

g.setFont(font);

g.setColor(Color.***RED***);

g.drawString("No Of Meals="+Integer.*toString*(philosopher.getNoMeals(2)), 920, 300);

g.setColor(Color.***BLACK***);

**if**(philosopher.getS(2)==4){

g.drawImage(state2,800 , 240, 100, 100, **null**);

g.drawString("Thinking", 920, 270);

}**else**{

g.drawImage(state1,800 , 240, 100, 100, **null**);

g.drawString("Eating", 920, 270);

}

}**public** **void** drawS3(Graphics g){

g.setFont(font);

g.setColor(Color.***RED***);

g.drawString("No Of Meals="+Integer.*toString*(philosopher.getNoMeals(3)), 920, 420);

g.setColor(Color.***BLACK***);

**if**(philosopher.getS(3)==4){

g.drawImage(state2,800, 360, 100, 100, **null**);

g.drawString("Thinking", 920, 390);

}**else**{

g.drawImage(state1,800, 360, 100, 100, **null**);

g.drawString("Eating", 920, 390);

}

}**public** **void** drawS4(Graphics g){

g.setFont(font);

g.setColor(Color.***RED***);

g.drawString("No Of Meals="+Integer.*toString*(philosopher.getNoMeals(4)), 920, 540);

g.setColor(Color.***BLACK***);

**if**(philosopher.getS(4)==4){

g.drawImage(state2,800, 480, 100, 100, **null**);

g.drawString("Thinking", 920, 510);

}**else**{

g.drawImage(state1,800, 480, 100, 100, **null**);

g.drawString("Eating", 920, 510);

}

}

**public** **void** drawQuote(Graphics g){

**if**(philosopher.getQuote(0)==1){

}

**if**(philosopher.getQuote(0)==2){

}**if**(philosopher.getQuote(0)==3){

}

}

}

**- App.java**

**import** javax.swing.\*;

**public** **class** App {

JFrame f1;

Chopstick chopstick[];

Philosopher philosophers[];

**public** App() {

f1 = **new** JFrame();

f1.setTitle("DINING PHILOSOPHER SIMULATOR");

f1.setSize(1100, 640);

f1.setVisible(**true**);

f1.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

f1.setLocationRelativeTo(**null**);

f1.setResizable(**false**);

initializePos();

initializePhilosopher();

Display obj = **new** Display();

obj.setBounds(0, 0, 1040, 600);

Thread t1 = **new** Thread(obj);

t1.start();

**new** Thread(philosophers[0]).start();

**new** Thread(philosophers[1]).start();

**new** Thread(philosophers[2]).start();

**new** Thread(philosophers[3]).start();

**new** Thread(philosophers[4]).start();

f1.add(obj);

f1.validate();

}

**public** **void** initializePos() {

chopstick = **new** Chopstick[5];

chopstick[0] = **new** Chopstick(0, 280, 380);

chopstick[1] = **new** Chopstick(1, 250, 225);

chopstick[2] = **new** Chopstick(2, 400, 120);

chopstick[3] = **new** Chopstick(3, 540, 250);

chopstick[4] = **new** Chopstick(4, 450, 400);

}

**public** **void** initializePhilosopher() {

philosophers = **new** Philosopher[5];

philosophers[0] = **new** Philosopher(0, chopstick[0], chopstick[4]);

philosophers[1] = **new** Philosopher(1, chopstick[1], chopstick[0]);

philosophers[2] = **new** Philosopher(2, chopstick[2], chopstick[1]);

philosophers[3] = **new** Philosopher(3, chopstick[3], chopstick[2]);

philosophers[4] = **new** Philosopher(4, chopstick[4], chopstick[3]);

}

**public** **static** **void** main(String args[]) {

**new** App();

}

}

**2. Pipeline Client-Server**

**- ClientUI.java**

**package** chatPipeline;

**import** java.awt.Button;

**import** java.awt.FlowLayout;

**import** java.awt.Frame;

**import** java.awt.Label;

**import** java.awt.Panel;

**import** java.awt.TextArea;

**import** java.awt.TextField;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.WindowAdapter;

**import** java.awt.event.WindowEvent;

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.io.PipedInputStream;

**import** java.io.PipedOutputStream;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

**import** java.util.Scanner;

**public** **class** ClientUI **extends** Frame **implements** Runnable,ActionListener {

**public** **static** TextField *txtMessage*;

**public** **static** TextArea *taContent*;

**public** Button btnSend;

**boolean** isConnected;

**private** PipedOutputStream out = **new** PipedOutputStream();

**private** PipedInputStream in = **new** PipedInputStream();

**static** String *msgOut*;

String tem;

**boolean** isSent= **false**;

**public** ClientUI(String clientTitle,PipedOutputStream out,PipedInputStream in) **throws** Exception {

**super**(clientTitle);

**this**.out = out;

**this**.in= in;

System.***out***.println("Client creation");

Panel panelTop = **new** Panel();

panelTop.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));

Panel panelContent = **new** Panel();

*taContent* = **new** TextArea(15, 35);

*taContent*.setEditable(**false**);

panelContent.add(*taContent*);

Panel panelBottom = **new** Panel();

panelBottom.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));

*txtMessage* = **new** TextField(25);

btnSend = **new** Button("Send");

panelBottom.add(*txtMessage*);

panelBottom.add(btnSend);

add(panelTop, "North");

add(panelContent);

add(panelBottom, "South");

addEventForButton();

setVisible(**true**);

pack();

addWindowListener(**new** WindowAdapter() {

**public** **void** windowClosing(WindowEvent we) {

System.*exit*(0);

}

});

setResizable(**false**);

}

**public** **void** addEventForButton() {

btnSend.addActionListener(**this**);

}

**public** **void** sendMessageToServer() {

*msgOut* = *txtMessage*.getText()+"";

*pushMsgToScreen*(*msgOut*);

*txtMessage*.setText("");

}

**public** **static** **void** pushMsgToScreen(String msg) {

*taContent*.append("\nClient: " + msg);

}

**public** **static** **void** receiveDataFromServer(String msg) {

*taContent*.append("\nServer: "+msg );

}

@Override

**public** **void** actionPerformed(ActionEvent e) {

**if** (e.getActionCommand() == "Send") {

isSent = **true**;

**try** {

writeMessage(*txtMessage*.getText());

sendMessageToServer();

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

}

//----------------------------------------------

@Override

**public** **void** run() {

**while**(**true**){

readMessage();

}

}

**private** **void** writeMessage(String tem) {

**try** {

**if**(isSent){

out.write(tem.getBytes());

}

isSent = **false**;

} **catch** (IOException e) {

e.printStackTrace();

}

}

**private** **void** readMessage() {

**byte**[] buf = **new** **byte**[1024];

**try** {

**int** len = in.read(buf);

System.***out***.println("Server: " + **new** String(buf, 0, 100));

*receiveDataFromServer*(**new** String(buf, 0, 100));

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

}

}

}

**- ServerUI.java**

**package** chatPipeline;

**import** java.awt.Button;

**import** java.awt.ComponentOrientation;

**import** java.awt.FlowLayout;

**import** java.awt.Font;

**import** java.awt.Frame;

**import** java.awt.Label;

**import** java.awt.Panel;

**import** java.awt.TextArea;

**import** java.awt.TextField;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.WindowAdapter;

**import** java.awt.event.WindowEvent;

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.io.PipedInputStream;

**import** java.io.PipedOutputStream;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

**import** java.util.Scanner;

**public** **class** ServerUI **extends** Frame **implements** Runnable,ActionListener {

**public** TextField txtMessage;

**public** **static** TextArea *taContent*;

**public** Button btnSend;

**boolean** isConnected;

**private** PipedInputStream in = **new** PipedInputStream();

**private** PipedOutputStream out = **new** PipedOutputStream();

**public** ServerUI(String clientTitle, PipedInputStream in,PipedOutputStream out) **throws** Exception{

**super**(clientTitle);

**this**.in = in;

**this**.out = out;

System.***out***.println("Server is starting...");

Panel panelTop = **new** Panel();

panelTop.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));

Panel panelContent = **new** Panel();

*taContent* = **new** TextArea(15,35);

*taContent*.setEditable(**false**);

*taContent*.setText("Server is started!!!\n--------------------------\n");

panelContent.add(*taContent*);

Panel panelBottom = **new** Panel();

panelBottom.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));

txtMessage = **new** TextField(25);

btnSend = **new** Button("Send");

panelBottom.add(txtMessage);

panelBottom.add(btnSend);

add(panelTop,"North");

add(panelContent);

add(panelBottom,"South");

addEventForButton();

setVisible(**true**);

pack();

addWindowListener(**new** WindowAdapter(){

**public** **void** windowClosing(WindowEvent we){

System.*exit*(0);

}

});

setResizable(**false**);

}

**public** **void** addEventForButton(){

btnSend.addActionListener(**this**);

}

**public** **void** sendMessageToClient() {

String msgOut = txtMessage.getText()+"";

pushMsgToScreen(msgOut);

txtMessage.setText("");

}

**public** **void** pushMsgToScreen(String msg){

*taContent*.setComponentOrientation(ComponentOrientation.***RIGHT\_TO\_LEFT***);

*taContent*.append("\nServer: "+msg);

}

**public** **void** receiveDataFromClient(String msg) {

*taContent*.append("\nClient: "+msg);

}

@Override

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getActionCommand()=="Send"){

**try** {

writeMessage(txtMessage.getText());

sendMessageToClient();

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

}

@Override

**public** **void** run() {

**while**(**true**){

readMessage();

}

}

**private** **void** readMessage() {

**byte**[] buf = **new** **byte**[1024];

**try** {

**int** len = in.read(buf);

System.***out***.println("Client: " + **new** String(buf, 0, 100));

receiveDataFromClient(**new** String(buf, 0, 100));

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

}

}

**private** **void** writeMessage(String tem) {

**try** {

out.write(tem.getBytes());

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**- PipedTest.java**

**package** chatPipeline;

**import** java.io.IOException;

**import** java.io.PipedInputStream;

**import** java.io.PipedOutputStream;

**public** **class** PipedTest {

**public** **static** **void** main(String[] args) **throws** Throwable {

PipedOutputStream c\_out = **new** PipedOutputStream();

PipedInputStream s\_in = **new** PipedInputStream();

PipedOutputStream s\_out = **new** PipedOutputStream();

PipedInputStream c\_in = **new** PipedInputStream();

Thread client = **new** Thread(**new** ClientUI("Client Chat!",c\_out,c\_in));

Thread server = **new** Thread(**new** ServerUI("Server Chat!",s\_in,s\_out));

**try** {

c\_out.connect(s\_in);

s\_out.connect(c\_in);

client.start();

server.start();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}