* Hệ thống phân tán gồm các máy tính độc lập không phụ thuộc lẫn nhau, có thể là các máy tính có kiến trúc khác nhau, có thể là các máy tính có phần mềm hệ thống khác nhau. Thông qua mạng máy tính, các phần mềm trên các máy tính khác nhau có khả năng phối hợp và chia sẻ tài nguyên. Nhằm thực hiện một nhiệm vụ chung, cung cấp dịch vụ một cách thống nhất. Người sử dụng không cần phải quan tâm tới các chi tiết của hệ thống.
* Ví dụ hệ thống phân tán: hệ thống www, hệ thống email.
* Một hệ thống phân tán luôn có các đặc trưng như chia sẻ tài nguyên, tính trong suốt và tính co giãn.
* Việc chia sẻ tài nguyên sẽ giảm chi phí, kết nối các tài nguyên, tăng tính sẵn sàng, hỗ trợ làm việc nhóm, tăng rủi ro về an toàn thông tin.
* Tính trong suốt có nghĩa là hệ thống là duy nhất với người sử dụng, trong suốt về quy mô và vị trí, che giấu tính phân tán của hệ phân tán.
* Có 8 loại trong suốt: truy cập, địa điểm, di trú, chuyển địa điểm, sao lưu, tương tranh, thứ lỗi và bền vững.
* Tính mở cho phép các thành phần có thể được sản xuất bởi các nhà sản xuất khác nhau.
* Tính co giãn – co giãn theo số lượng người sử dụng, không gian địa lý và tổ chức.
* Các thành phần của hệ phân tán:
* Phần cứng.
* Phần mềm (DOS – gắn với hệ thống phần cứng, NOS – gắn trên máy tính cục bộ, Middleware – cài đặt các dịch vụ cơ bản để thực hiện các ứng dụng).
* Phần mềm trung gian.

Kiến trúc hệ phân tán

Xem xét tổ chức của một hệ phân tán: Tổ chức logic gồm các thành phần mềm, cách thức kết nối, kiẻu dữ liệu trao đổi -> kiến trúc phần mềm. Thực thi vật lý cách thức xếp đặt/cài đặt thành phần phần mềm lên các thiết bị vật lý -> kiến trúc hệ thống.

Các kiến trúc thường dùng: kiến trúc phân tầng, kiến trúc hướng đối tượng, kiến trúc hướng dữ liệu, kiến trúc hướng sự kiện, kiến trúc Microservices.

Kiến trúc phân tầng: các chức năng trên hệ thống được phân rã thành các chức năng con.

* Các chức năng con được thực hiện bởi các module phần mềm.
* Các module phần mềm khác nhau trên cùng hệ thống phối hợp và tương tác với nhau để thực hiện cùng chức năng.
* Để đơn giản hệ thống cần giảm thiểu liên kết giữa các module.
* Ví dụ: mô hình OSI, mô hình Middleware.

Kiến trúc hướng đối tượng

* Ưu: Ánh xạ vào các đối tượng trong thế giới thật, dễ dàng bảo trì và nâng cấp, tính tái sử dụng, kiểm soát lỗi, mở rộng chức năng mà không ảnh hưởng hệ thống, dễ dàng kiểm thử, giảm thời gian và chi phí phát triển.
* Nhược: Khó khăn trong việc xác định các lớp, các đối tượng, kích cỡ chương trình lớn, chương trình chạy chậm hơn, không phù hợp với mọi bài toán.

Kiến trúc hướng sự kiện

* Thành phần hệ thống trao đổi thông tin với nhau qua các sự kiện.
* Các sự kiện chứa các thông tin cần trao đổi.
* Các sự kiện có thể kích hoạt các thao tác trong tiến trình.
* Có thể thực hiện theo mô hình point-point hoặc mô hình trục quảng bá sự kiện
* Ví dụ: mô hình thuê bao/xuất bản.

Kiến trúc hướng dữ liệu

* Các thành phần trao đổi thông tin thông qua kho dữ liệu.

Kiến trúc hướng dịch vụ Microservices

* Xây dựng ứng dụng dựa trên số lượng nhỏ các services, mỗi service chạy trên tiến trình riêng và hoàn toàn triển khai dộc lập được.
* Ưu: đơn giản triển khai, đơn giản để hiểu, tái sử dụng, nhanh chóng cách ly thành phần hỏng, giảm thiểu nguy cơ khi thực hiện thay đổi.

Container hóa là một phương pháp ảo hóa mức hệ điều hành để triển khai và chạy các ứng dụng phân tán mà không cần khởi chạy toàn bộ máy ảo cho mỗi ứng dụng.

Nó cho phép nhiều ứng dụng hoặc dịch vụ riêng biệt chạy trên một host duy nhất và truy cập và nhân hệ điều hành của host đó.

Container bao gồm toàn bộ môi trường runtime của một ứng dụng/dịch vụ, cộng với tất cả các gói phụ thuộc, các thư viện và các tệp binary khác của nó, cùng với các tệp cấu hình cần thiết để chạy nó, tất cả chúng được đóng thành một gói.

Thời đại triển khai Container:

* Các Container như VM, nhưng chúng có tình cô lập để chia sẽ HĐH giữa các ứng dụng. Do đó, container được coi là nhẹ.
* Tương tự như VM, một container có hệ thống tệp, CPU, bộ nhớ, process space.
* Khi chúng được tách rời khỏi cơ sở hạ tầng bên dưới, chúng có thể khả chuyển trên cloud hoặc các bản phân phối HĐH.

Container orchestration là các công cụ, dịch vụ dùng để điều phối và quản lý nhiều contaier sao cho chúng làm việc một cách hiệu quả nhất. Ví dụ: EC2, ACS, Cloud Foundry’s Diego, CoreOS Fleet, Docker Swarm, Kubernetes.

Kubernetes giúp quản lý các container chạy các ứng dụng và đảm bảo rằng không có khoảng thời gian downtime. Kubernetes cung cấp một framework để chạy các hệ phân tán một cách mạnh mẽ. Nó đảm nhiệm việc nhân rộng và chuyển đổi dự phòng các ứng dụng.

Có chức năng như:

* service discovery và cân bằng tải
* điều phối bộ nhớ
* tự động rollouts và rollbacks
* đóng gói tự động
* tự phục hồi
* quản lý cấu hình và bảo mật

Kiến trúc hệ thống

* Kiến trúc tập trung: kiến trúc client-server, phân tầng ứng dụng, kiến trúc đa tầng và software agent.
* Kiến trúc không tập trung
* Kiến trúc hỗn hợp

Fog Computing là một trong những giải pháp mở rộng của cloud computing. Dữ liệu và các ứng dụng được lưu trữ giữa nguồn dữ liệu và đám mây. Vì là một cơ sở hạ tầng điện toán phi tập trung, Fog Computing giúp thu hẹp khoảng cách giữa đám mây với nơi dữ liệu được tạo ra và hoạt động.

Fog Computing hoạt động thông qua các thiết bị cục bộ. Trong khi đó, đèn hiệu IoT sẽ thu thập các ữ liệu cần thiết. Những dữ liệu này tiếp tục được đưa đến một nút sương mù gần với nguồn dữ liệu. Chúng sẽ được phân tích, xử lý bởi các nút sương mù và gửi lên đám mây để lưu trữ lâu dài khi cần thiết. Một số loại thiết bị cục bộ thường được sử dụng là: máy ảnh, bộ định tuyến, công tắc, máy chủ nhúng, bộ điều khiển.

Ưu điểm của Fog Computing: tính linh hoạt, phân tích dữ liệu theo thời gian thực, hạn chế độ trễ, làm giảm băng thông và phân bổ tốt hơn.

Tiến trình là chương trình đang hoạt động (Processor, memory, IO, files)

Quá trình tạo tiến trình:

* Định danh tiến trình
* Trạng thái của tiến trình
* Ngữ cảnh của tiến trình (trạng thái CPU, bộ xử lý, bộ nhớ chính, tài nguyên sử dụng, tài nguyên tạo lập)
* Thông tin giao tiếp (tiến trình cha, tiến trình con, độ ưu tiên
* Tiến trình thống kê (CPU time, time chờ)

Luồng là một luồng thực thi của tiến trình. Tiến trình có nhiều luồng thực thi còn gọi là tiến trình đa luồng. Các luồng của tiến trình dùng môi trường thực hiện chung của tiến trình. Trao đổi thông tin giữa các luồng thông qua các biến chia sẻ.

Lợi ích của tiến trình đa luồng:

* Không bị dừng chương trình với các lời gọi hệ thống dừng
* Tính toán song song

Server đơn luồng chỉ xử lý được một yêu cầu tại một thời điểm, các yêu cầu có thể được xử lý tuần tự, các yêu cầu có thể được xử lý bởi các tiến trình khác nhau, không đảm bảo tính trong suốt.

Client đa luồng tách biệt giao diện người sử dụng và xử lý. Giải quyết các vấn đề các thao tác chờ đợi lẫn nhau. Tăng tốc độ khi làm việc với nhiều server khác nhau. Che giấu các chi tiết cài đặt.

Phải di trú mã vì nó tăng hiệu năng, giảm thiểu các trao đổi thông tin trên mạng giữa client và server.

Mobile agent là những thành phần phần mềm có thể tự di chuyển từ nơi này sang nơi khác.

Lời gọi thủ tục từ xa

Giao thức yêu cầu – trả lời là cơ chế bậc cao hơn truyền thông điệp, cho phép trao đổi thông tin giữa 2 tiến trình bằng 2 thông báo gửi nhận liên tiếp. Hỗ trợ các lời gọi từ xa. Đồng bộ và tin cậy.

Yêu cầu – trả lời có đặc điểm không cẩn báo cáo và không cần kiểm soát luồng.

Liên kết Client-Server:

* Liên kết cục bộ dịch vụ cục bộ trên máy tính server cho biết có những thủ tục nào được cung cấp.
* Liên kết toàn cục dịch vụ thư mục cho biết địa chỉ máy tính và vị trí của dịch vụ.

Akamai là một nhà cung cấp dịch vụ phân phối nội dung và điện toán đám mây của Mỹ. Mạng phân phối nội dung của Akamai là một trong những nền tảng điện toán phân toán lớn nhất thế giới, chịu trách nhiệm phục vụ từ 15 đến 30 phần trăm lưu lượng web.

Conit

A screenshot of a calculator

Description automatically generated

Vector clock -> thời gian thực hiện

Order deviation -> sai lệch về thứ tự thực hiện

Numerical deviation -> sai lệch về giá trị

Kích thước lớn: các bản sao sẽ bị rơi vào trạng thái không thống nhất

Kích thước nhỏ: số lượng conit nhiều -> quản lý phức tạp

Với các quá trình thực hiện khác nhau, tất cả các tiến trình luôn luôn cho một kết quả

Các thao tác trên dữ liệu:

* Đọc (Ri(x)b)
* Ghi (Wi(x)a)
* Giá trị khởi tạo của các dữ liệu là NIL

Thống nhất tuần tự:

* Các tiến trình đều có một chuỗi thao tác cục bộ
* Các thao tác cục bộ của các tiến trình được tổng hợp thành các thứ tự thực hiện các thao tác trên kho dữ liệu
* Có thể có các thứ tự thực hiện khác nhau trên kho dữ liệu
* Điều kiện của thống nhất tuần tự là các kết quả luôn luôn như nhau
* Tất cả các tiến trình đều nhìn thấy một thứ tự của các thao tác ghi

Thống nhất (consistency): áp dụng cho tập hợp phần tử dữ liệu.

Phù hợp (coherence): áp dụng cho một đơn vị dữ liệu/phần tử dữ liệu.

Mô hình hướng client:

* Eventual Consistency (thống nhất cuối cùng) chủ yếu các tiến trình thực hiện đọc. Rất ít các tiến trình thực hiện cập nhật (DNS, WWW). Xung đột ghi-ghi hầu như không xảy ra. Xem xét xung đột đọc ghi. Nếu dữ liệu không bị thay đổi trong thời gian đủ dài.

Mô hình thống nhất hướng client:

* Cung cấp đảm bảo thống nhất cho các truy cập của một client đơn vào kho dữ liệu. Các kiểu mô hình: đọc đơn điệu, ghi đơn điệu, đọc ghi kết quả đã ghi, ghi theo thao tác đã đọc.

Ký hiệu:

* Li: bản sao thứ i
* Xi[t]: phần tử dữ liệu x ở bản sao cục bộ Li, thời điểm t
* WS(xi[t1]; xj[t2]): tất cả các thao tác WS(xi[t1]) đã được phổ biến đến bản sao Lj, sau khoảng thời gian t2.

Đọc dữ liệu ghi trên một tiến trình nếu thao tác đọc xảy ra sau thao tác ghi, thao tác đọc sẽ xảy ra sau khi thao tấc ghi hoàn thành. Các thao tác đọc sẽ chờ sau khi thao tác ghi hoàn thành mới thực hiện. Ví dụ: cập nhật trang web, đọc nội dung trang web, cập nhật mật khẩu.

Ghi sau khi đọc là thao tác ghi thực hiện sau thao tác đọc, thao tác ghi chỉ được thực hiện sau khi thao tác đọc hoàn thành. Ví dụ chỉ có thể trả lời sau khi đọc nội dung thư.

Trạng thái/thao tác:

* Giải pháp cập nhật dữ liệu: chỉ thông báo có cập nhật nhằm giảm thông tin cần truyền. thích hợp cho trường hợp ghi nhiều – đọc ít
* Truyền dữ liệu cập nhật: thích hợp cho trường hợp đọc nhiều ghi ít
* Truyền thao tác cập nhật: các bản sao cần theo dõi trạng thái của dữ liệu, chỉ cần truyền các tham số cần thiết cho thao tác cập nhật
* Trạng thái nhiều thì thao tác ít và ngược lại.

Push: server sau khi cập nhật dữ liệu thông báo cho tất cả các client

Pull: client khi cần dữ liệu sẽ hỏi server

Multicasting: thích hợp trong trường hợp 1 bản sao muốn quảng bá cho các bản sao khác trong 1 data store. Phù hợp với push-based

Unicasting: phù hợp với pull-based

Tiến trình cộng và tiến trình trừ

private static int count = 0;

private static object lockObject = new object();

public static void TieuTrinhCong()

{

//luu tru du lieu thoi gian vao bien startTime

DateTime startTime = DateTime.Now;

//chay mot vong lap 2500 lan

for (int i = 0; i < 2500; i++)

{

//dung mot khoa 'lockobject' de dam bao thuc hien chi co mot luong tai mot thoi diem

lock (lockObject)

{

//tang bien count va in ra thong bao thoi gian hien tai

count++;

Console.WriteLine($"({DateTime.Now:HH:mm:ss.ffff}) TieuTrinhCong: count = {count}");

}

//dung 1 ml giay

Thread.Sleep(1);

}

//luu thoi gian hien tai va luu tring endTime

DateTime endTime = DateTime.Now;

//in ra thong bao chi ra tong thoi gian thuc thi

Console.WriteLine($"TieuTrinhCong: {(endTime - startTime).TotalMilliseconds} ");

}

public static void TieuTrinhTru()

{

//luu tru du lieu thoi gian vao bien startTime

DateTime startTime = DateTime.Now;

//chay mot vong lap 2500 lan

for (int i = 0; i < 2500; i++)

{

//dung mot khoa 'lockobject' de dam bao thuc hien chi co mot luong tai mot thoi diem

lock (lockObject)

{

//giam bien count va in ra thong bao thoi gian hien tai

count--;

Console.WriteLine($"({DateTime.Now:HH:mm:ss.ffff}) TieuTrinhTru: count = {count} ");

}

//dung 1 ml giay

Thread.Sleep(1);

}

//luu thoi gian hien tai va luu tring endTime

DateTime endTime = DateTime.Now;

//in ra thong bao chi ra tong thoi gian thuc thi

Console.WriteLine($"TieuTrinhTru: {(endTime - startTime).TotalMilliseconds} ");

}

static void Main(string[] args)

{//in thong bao thoi gian cac luong thuc thi

Console.WriteLine("{0} : Starting threads...", DateTime.Now.ToString("HH:mm:ss.ffff"));

//tao ra mot luong moi de thuc thi TieuTrinhCong

Thread threadCong = new Thread(TieuTrinhCong);

//tao ra mot luong moi de thuc thi TieuTrinhTru

Thread threadTru = new Thread(TieuTrinhTru);

//bat dau luong threadCong

threadCong.Start();

//bat dau luong threadTru

threadTru.Start();

//luong chinh cho threadCong hoan thanh

threadCong.Join();

//luong chinh cho threadTru hoan thanh

threadTru.Join();

//in thong bao chi thoi gian hoan thanh luong chinh

Console.WriteLine("{0} : Main thread complete. Press Enter.", DateTime.Now.ToString("HH:mm:ss.ffff"));

Console.ReadLine();

}

Điều khiển trạng thái

public static void DisplayMessage1()

{

// Lặp đi lặp lại việc hiển thị một thông báo ra cửa sổ Console.

while (true)

{

try

{

Console.WriteLine("{0} : Second thread running. Enter" + " (S)uspend, (R)esume,(I)nterrupt, or(E)xit.", DateTime.Now.ToString("HH: mm:ss.ffff"));

// Nghỉ 2 giây.

Thread.Sleep(2000);

}

catch (ThreadInterruptedException)

{

// Tiểu trình đã bị gián đoạn. Việc bắt ngoại lệ ThreadInterruptedException cho phép ví dụ này

// thực hiện hành động phù hợp và tiếp tục thực thi.

Console.WriteLine("{0} : Second thread interrupted.",

DateTime.Now.ToString("HH:mm:ss.ffff"));

}

catch (ThreadAbortException abortEx)

{

// Đối tượng trong thuộc tính ThreadAbortException.ExceptionState được cung cấp

// bởi tiểu trình đã gọi Thread.Abort.

// Trong trường hợp này, nó chứa một chuỗi mô tả lý do của việc hủy bỏ.

Console.WriteLine("{0} : Second thread aborted ({1})",

DateTime.Now.ToString("HH:mm:ss.ffff"), abortEx.ExceptionState);

}

}

}

static void Main(string[] args)

{

Console.WriteLine(": GIAI THICH O CUOI DOAN CODE");

// Tạo một đối tượng Thread và truyền cho nó một thể hiện ủy nhiệm ThreadStart tham chiếu

// đến DisplayMessage.

Thread thread = new Thread(new ThreadStart(DisplayMessage1));

Console.WriteLine("{0} : Starting second thread.", DateTime.Now.ToString("HH:mm:ss.ffff"));

// Khởi chạy tiểu trình thứ hai.

thread.Start();

// Lặp và xử lý lệnh do người dùng nhập.

char command = ' ';

do

{

string input = Console.ReadLine();

if (input.Length > 0)

command = input.ToUpper()[0];

else command = ' ';

switch (command)

{

case 'S': // Tạm hoãn tiểu trình thứ hai.

Console.WriteLine("{0} : Suspending second thread.",

DateTime.Now.ToString("HH:mm:ss.ffff"));

thread.Suspend();

break;

case 'R': // Phục hồi tiểu trình thứ hai.

try

{

Console.WriteLine("{0} : Resuming second thread.",

DateTime.Now.ToString("HH:mm:ss.ffff"));

thread.Resume();

}

catch (ThreadStateException)

{

Console.WriteLine("{0} : Thread wasn't suspended.",

DateTime.Now.ToString("HH:mm:ss.ffff"));

}

break;

case 'I': // Gián đoạn tiểu trình thứ hai.

Console.WriteLine("{0} : Interrupting second thread.",

DateTime.Now.ToString("HH:mm:ss.ffff"));

thread.Interrupt();

break;

case 'E':

// Hủy bỏ tiểu trình thứ hai và truyền một đối tượng trạng thái cho tiểu trình đang bị hủy,

// trong trường hợp này là một thông báo.

Console.WriteLine("{0} : Aborting second thread.",

DateTime.Now.ToString("HH:mm:ss.ffff"));

thread.Abort("Terminating example.");

// Đợi tiểu trình thứ hai kết thúc.

thread.Join();

break;

}

} while (command != 'E');

// Nhấn Enter để kết thúc.

Console.WriteLine("Main method complete. Press Enter.");

Console.ReadLine();

}

Nhân 2 ma trận

static void Main(string[] args)

{

int rows = 3;

int cols = 3;

int[,] matrix1 = new int[rows, cols];

int[,] matrix2 = new int[rows, cols];

Console.WriteLine("Nhap ma tran thu nhat (3x3): ");

ReadMatrix(matrix1);

Console.WriteLine("Nhap ma tran thu hai (3x3): ");

ReadMatrix(matrix2);

int[,] resultMatrix = new int[rows, cols];

MultiplyMatrices(matrix1, matrix2, resultMatrix);

Console.WriteLine("Ket qua: ");

PrintMatrix(resultMatrix);

Console.WriteLine("Nhan phim Enter de thoat chuong trinh");

Console.ReadLine();

}

static void ReadMatrix(int[,] matrix)

{

for (int i = 0; i < matrix.GetLength(0); i++)

{

Console.WriteLine($"Nhap dong {i + 1}:");

string[] input = Console.ReadLine().Split(' ');

if (input.Length != matrix.GetLength(1))

{

Console.WriteLine("Khong hop le");

i--;

continue;

}

for (int j = 0; j < matrix.GetLength(1); j++)

{

matrix[i, j] = int.Parse(input[j]);

}

}

}

static void PrintMatrix(int[,] matrix)

{

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

Console.Write(matrix[i, j] + " ");

}

Console.WriteLine();

}

}

static void MultiplyMatrices(int[,] matrix1, int[,] matrix2, int[,] resultMatrix)

{

if (matrix1.GetLength(1) != matrix2.GetLength(0))

{

Console.WriteLine("Ma tran khong hop le");

return;

}

Task[] tasks = new Task[resultMatrix.GetLength(0) \* resultMatrix.GetLength(1)];

int taskIndex = 0;

for (int i = 0; i < resultMatrix.GetLength(0); i++)

{

for (int j = 0; j < resultMatrix.GetLength(1); j++)

{

tasks[taskIndex++] = Task.Run(() =>

{

int sum = 0;

for (int k = 0; k < matrix1.GetLength(1); k++)

{

sum += matrix1[i, k] \* matrix2[k, j];

}

resultMatrix[i, j] = sum;

});

}

}

Task.WaitAll(tasks);

}

Sắp xếp chẵn lẻ

static void Main(string[] args)

{

Console.WriteLine("Nhap day so va cach nhau bang dau cach (Space): ");

string input = Console.ReadLine();

try

{

int[] numbers = input.Split(' ').Select(int.Parse).ToArray();

Task<List<int>> evenTask = Task.Run(() => numbers.Where(x => x % 2 == 0).OrderBy(x => x).ToList());

Task<List<int>> oddTask = Task.Run(() => numbers.Where(x => x % 2 != 0).OrderBy(x => x).ToList());

Task.WaitAll(evenTask, oddTask);

List<int> sortedNumbers = evenTask.Result.Concat(oddTask.Result).ToList();

Console.WriteLine("Ket qua:");

Console.WriteLine(string.Join(", ", sortedNumbers));

}

catch (FormatException)

{

Console.WriteLine("Nhap day so va cach nhau bang dau cach (Space): ");

}

}

Tính số Pi

static void Main(string[] args)

{

int numThreads = 4;

long numTerms = 100000000;

long termsPerThread = numTerms / numThreads;

Task<double>[] tasks = new Task<double>[numThreads];

for (int i = 0; i < numThreads; i++)

{

long startTerm = i \* termsPerThread;

long endTerm = (i == numThreads - 1) ? numTerms : startTerm + termsPerThread;

tasks[i] = Task.Run(() => TinhSoPi(startTerm, endTerm));

}

Task.WaitAll(tasks);

double pi = 0;

for (int i = 0; i < numThreads; i++)

{

pi += tasks[i].Result;

}

Console.WriteLine("Ket qua: " + pi);

Console.WriteLine("Nhan Enter de thoat");

Console.ReadLine();

}

static double TinhSoPi(long startTerm, long endTerm)

{

double sum = 0;

for (long i = startTerm; i < endTerm; i++)

{

sum += (Math.Pow(-1, i) / (2 \* i + 1));

}

return 4 \* sum;

}

Server

static void Main(string[] args)

{

//tao ket noi voi dia chi IP localhost voi port la 2010

IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2010);

//tao ket noi TCP

Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

//Server mo ket noi va bat dau lang nghe

server.Bind(iep);

server.Listen(10);

Console.WriteLine("Cho ket noi tu client");

//khi co Client gui yeu cau thi se ket noi

Socket client = server.Accept();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//tao ket noi voi Client

NetworkStream ns = new NetworkStream(client);

//cho phep doc tin nhan tu Client

StreamReader sr = new StreamReader(ns);

//cho phep gui tin nhan toi Client

StreamWriter sw = new StreamWriter(ns);

Console.WriteLine("Da ket noi voi: {0}", client.RemoteEndPoint.ToString());

string s = "Chao ban den voi Server";

sw.WriteLine(s);

sw.Flush();

string input;

//tao mot vong lap ket noi

while (true)

{

Console.Write("Gui tin toi Client: ");

input = Console.ReadLine();

//Neu chuoi nhan duoc la Thoat thi thoat

if (input.ToUpper().Equals("THOAT")) break;

//Gui tra lai cho server chuoi s

s = sr.ReadLine();

Console.WriteLine("Client gui toi:{0}", s);

}

//khi tat ket noi thi tat ca Server va Client

client.Shutdown(SocketShutdown.Both);

client.Close();

server.Close();

}

Client

static void Main(string[] args)

{

//tao ket noi voi dia chi IP localhost voi port la 2010

IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2010);

//tao ket noi TCP

Socket client = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

//Client se gui yeu cau ket noi

client.Connect(iep);

//tao ket noi voi Server

NetworkStream ns = new NetworkStream(client);

//cho phep doc tin nhan tu Server

StreamReader sr = new StreamReader(ns);

//cho phep gui tin nhan toi Server

StreamWriter sw = new StreamWriter(ns);

string s = sr.ReadLine();

Console.WriteLine("Server da gui:{0}", s);

string input;

//tao mot vong lap ket noi

while (true)

{

Console.Write("Gui tin toi Server: ");

input = Console.ReadLine();

sw.WriteLine(input);

sw.Flush();

//Neu chuoi nhan duoc la Thoat thi thoat

if (input.ToUpper().Equals("THOAT")) break;

//Gui tra lai cho client chuoi s

s = sr.ReadLine();

Console.WriteLine("Server gui:{0}", s);

}

//khi tat ket noi thi tat Client

client.Disconnect(true);

client.Close();

}

Chat Client Server

class ChatServer

{

static void Main(string[] args)

{

IPEndPoint iep = new IPEndPoint(IPAddress.Any, 2010);

TcpListener listener = new TcpListener(iep);

listener.Start();

Console.WriteLine("Server đang cho ket noi...");

TcpClient client = listener.AcceptTcpClient();

Console.WriteLine("Client da ket noi!");

NetworkStream ns = client.GetStream();

StreamReader sr = new StreamReader(ns);

StreamWriter sw = new StreamWriter(ns);

sw.AutoFlush = true;

Thread receiveThread = new Thread(() => ReceiveMessages(sr));

receiveThread.Start();

string message;

while (true)

{

message = Console.ReadLine();

sw.WriteLine("Server: " + message);

if (message.ToLower() == "exit") break;

}

receiveThread.Join();

sr.Close();

sw.Close();

ns.Close();

client.Close();

listener.Stop();

}

static void ReceiveMessages(StreamReader sr)

{

string message;

while (true)

{

try

{

message = sr.ReadLine();

if (message == null) break;

Console.WriteLine(message);

if (message.ToLower() == "Client: exit") break;

}

catch (Exception)

{

break;

}

}

}

static void Main(string[] args)

{

TcpClient client = new TcpClient("10.0.242.84", 2010);

Console.WriteLine("Da ket noi server!");

NetworkStream ns = client.GetStream();

StreamReader sr = new StreamReader(ns);

StreamWriter sw = new StreamWriter(ns);

sw.AutoFlush = true;

Thread receiveThread = new Thread(() => ReceiveMessages(sr));

receiveThread.Start();

string message;

while (true)

{

message = Console.ReadLine();

sw.WriteLine("Client: " + message);

if (message.ToLower() == "exit") break;

}

receiveThread.Join();

sr.Close();

sw.Close();

ns.Close();

client.Close();

}

static void ReceiveMessages(StreamReader sr)

{

string message;

while (true)

{

try

{

message = sr.ReadLine();

if (message == null) break;

Console.WriteLine(message);

if (message.ToLower() == "Server: exit") break;

}

catch (Exception)

{

break;

}

}

}