Câu 1:

a. 7 tầng của mô hình OSI và chức năng

Tầng	Tên tầng	Chức năng chính
7	**Application**	Giao tiếp với phần mềm người dùng; cung cấp dịch vụ mạng cho ứng dụng (ví dụ: HTTP, FTP)
6	**Presentation**	Biến đổi dữ liệu: mã hóa, nén, chuyển đổi định dạng (ví dụ: chuyển đổi giữa ASCII và EBCDIC)
5	**Session**	Thiết lập, quản lý và kết thúc phiên giao tiếp giữa hai thiết bị
4	**Transport**	Đảm bảo truyền dữ liệu chính xác (TCP – có kiểm soát lỗi; UDP – không)
3	**Network**	Định tuyến và định địa chỉ (IP); xử lý phân mảnh, tái cấu trúc gói tin
2	**Data Link**	Truyền dữ liệu giữa 2 nút kề nhau; kiểm soát lỗi đường truyền (MAC, Ethernet)
1	**Physical**	Giao tiếp vật lý: tín hiệu điện, ánh sáng, tần số, dây cáp, đầu nối

b. Cơ chế "Encapsulation" và "Decapsulation"

Encapsulation – Bao đóng:

- Khi **dữ liệu đi từ tầng 7 → tầng 1**, mỗi tầng **thêm một header** riêng (gồm thông tin điều khiển cần thiết).
- Tầng thấp hơn **bọc** dữ liệu của tầng trên trong **một đơn vị dữ liệu mới**.

Decapsulation – Bóc tách:

- Ở phía nhận, dữ liệu đi **từ tầng 1 → tầng 7** và **mỗi tầng loại bỏ header tương ứng** để xử lý thông tin.

ql Ví dụ về các thông tin điều khiển thêm vào ở mỗi tầng:

Tầng	Header chứa gì?
Transport (TCP)	Port nguồn, port đích, số thứ tự
Network (IP)	Địa chỉ IP nguồn/đích, TTL
Data Link (Ethernet)	MAC nguồn/đích, CRC

a. Tại sao phải phân mảnh?
   - Một số tầng dưới (như Data Link) hạn chế kích thước khung truyền (gọi là MTU – Maximum Transmission Unit).
   - Khi gói tin lớn hơn MTU, cần chia nhỏ (fragment) ở tầng Network hoặc Data Link để phù hợp với giới hạn này.

b – Quá trình phân mảnh và tái cấu trúc

Tại Sender (bên gửi):

1. IP kiểm tra kích thước gói tin.
2. Nếu vượt MTU → chia thành các fragment nhỏ hơn.
3. Gán identification (ID giống nhau), offset, và MF (More Fragments) để đánh dấu thứ tự và sự tồn tại của phần tiếp theo.

Tại Receiver (bên nhận):

1. Nhận các fragment.
2. Dựa vào ID, offset, MF để lắp lại đúng thứ tự.
3. Sau khi đầy đủ → chuyển lên tầng trên để xử lý.

c – Ảnh hưởng của phân mảnh

Ảnh hưởng	Mô tả
Giảm hiệu năng	Quá nhiều fragment → tăng overhead và độ trễ
Giảm độ tin cậy	Nếu 1 mảnh bị mất → toàn bộ gói bị hỏng
Linh hoạt	Cho phép truyền qua các mạng có MTU khác nhau

Câu 2:

a) Định nghĩa và vị trí middleware trong ngăn xếp giao thức

Định nghĩa:

Middleware là phần mềm nằm giữa hệ điều hành và các ứng dụng, giúp các ứng dụng phân tán giao tiếp, chia sẻ dữ liệu và dịch vụ với nhau một cách dễ dàng, minh bạch và hiệu quả.

Nói đơn giản: Middleware là "cầu nối" giữa các phần mềm ứng dụng chạy trên các hệ thống khác nhau.

Vị trí trong mô hình giao thức:

Mô hình	Vị trí Middleware
OSI	Trên tầng Application (tầng 7)
TCP/IP	Trên tầng Application (thậm chí nằm trên cả giao thức HTTP, FTP...)

Middleware hoạt động trên các giao thức mạng, nhưng dưới các ứng dụng cuối (như hệ thống đặt vé, ngân hàng, v.v.)

b) Phân loại Middleware và ví dụ

Loại Middleware	Mô tả	Ví dụ
Mở hoàn toàn (fully open)	Có thể truy cập mã nguồn, được tiêu chuẩn hóa, hỗ trợ nhiều nền tảng	CORBA (Common Object Request Broker Architecture)
Giao diện mở (open interface)	Cung cấp API mở, nhưng không bắt buộc mở mã nguồn	ODBC (Open Database Connectivity)
Giao thức mở (open protocol)	Cho phép các hệ thống khác nhau giao tiếp qua giao thức chuẩn	DRDA (Distributed Relational Database Architecture của IBM)
Middleware riêng (proprietary)	Chỉ hoạt động trên nền tảng cụ thể, không tiêu chuẩn hóa	ActiveX, OLE (Object Linking and Embedding – của Microsoft)

c) Vai trò của Middleware trong hệ thống phân tán

Chức năng chính:

1. Giao tiếp giữa tiến trình từ xa
   → Ví dụ: RPC (Remote Procedure Call), RMI (Remote Method Invocation)
2. Ẩn tính phức tạp của mạng (Transparency)
   → Người lập trình không cần biết chi tiết về IP, định tuyến, hay cách kết nối vật lý.
3. Quản lý dữ liệu và tài nguyên phân tán
   → Tạo cơ chế đồng bộ, khóa tài nguyên, quản lý truy cập
4. Tạo giao tiếp không đồng bộ hoặc theo sự kiện
   → Sử dụng mô hình message queue, publish-subscribe
5. Tích hợp các hệ thống khác nhau
   → Cho phép các ứng dụng cũ và mới (legacy systems) làm việc cùng nhau.

Ví dụ:

- Một ứng dụng ngân hàng chạy ở TP.HCM muốn gọi một dịch vụ xử lý giao dịch ở Hà Nội → middleware sẽ lo việc gọi từ xa, truyền dữ liệu và nhận kết quả, mà ứng dụng không cần xử lý chi tiết mạng.

Câu 3 :

a) Định nghĩa “thông điệp” và “chuyển thông điệp”

Thông điệp (Message)

Là **đơn vị dữ liệu** được trao đổi giữa hai tiến trình (process) trong hệ thống phân tán.
Một thông điệp có thể chứa dữ liệu, địa chỉ đích, mã lệnh hoặc các thông tin điều khiển.

Ví dụ: Một thông điệp gửi từ tiến trình A đến B có thể chứa lệnh xử lý đơn hàng, thông tin người dùng, v.v.

Chuyển thông điệp (Message Passing)

Là **cơ chế giao tiếp chính giữa các tiến trình trong hệ thống phân tán**, khi **không có vùng nhớ chung**.
Tiến trình gửi (sender) tạo thông điệp và gửi qua mạng; tiến trình nhận (receiver) nhận và xử lý thông điệp đó.

Gồm 2 thao tác cơ bản:

- `Send(destination, message)`
- `Receive(source, message)`

b) Ưu nhược điểm của phương pháp chuyển thông điệp

(So với **chia sẻ bộ nhớ** – shared memory)

Tiêu chí	Message Passing	Shared Memory
Tính trong suốt (Transparency)	Tốt hơn trong môi trường phân tán (các tiến trình cách xa nhau, không cần chia sẻ bộ nhớ vật lý).	Chỉ phù hợp với hệ thống đa tiến trình trên cùng máy. Không thể dùng giữa các máy khác nhau.
Độ tin cậy (Reliability)	Có thể thiết kế cơ chế kiểm lỗi (timeout, ack). Tuy nhiên, dễ bị lỗi mạng gây mất thông điệp.	Tin cậy hơn nếu trên cùng hệ thống vật lý. Nhưng dễ bị lỗi do xung đột truy cập (race condition).
Hiệu năng (Performance)	Chậm hơn vì phải đóng gói, gửi/nhận qua mạng.	Nhanh hơn do truy cập trực tiếp vào bộ nhớ chung.

Câu 4 :

a) Mục đích và bối cảnh ra đời của MPI

Mục đích của MPI (Message Passing Interface):

MPI là **giao diện lập trình chuẩn** cho việc **truyền thông điệp giữa các tiến trình** trong **hệ thống tính toán song song và phân tán**.

Nó giúp các tiến trình:

- **Giao tiếp qua mạng** (thường không có bộ nhớ chung)
- **Trao đổi dữ liệu hiệu quả**
- **Viết mã chương trình di động và tối ưu hiệu năng cao**

Bối cảnh ra đời:

- Trước MPI, có nhiều thư viện truyền thông điệp khác nhau, thiếu chuẩn chung.
- MPI ra đời năm **1994**, do diễn đàn MPI (MPI Forum) phát triển, với mục tiêu:
    - Đưa ra **chuẩn mở**, **hiệu năng cao**
    - **Độc lập nền tảng** và **ngôn ngữ lập trình**
    - Dùng nhiều trong **siêu máy tính, HPC, cụm máy (clusters)**

b) 4 hàm nguyên thủy cơ bản trong MPI

Hàm	Loại	Tính năng

1. `MPI_Send(...)`

- **Đồng bộ (Blocking)**, **chuẩn (standard mode)**
- Gửi thông điệp và **chờ cho đến khi buffer dữ liệu an toàn để sử dụng lại**
- Có thể **đợi đến khi nhận đã sẵn sàng hoặc hệ thống copy xong vào buffer nội bộ**
- **Không đảm bảo tuyệt đối về thời điểm gửi thực sự**

2. `MPI_Bsend(...)` (Buffered Send)

- **Đồng bộ**, nhưng **dùng buffer người dùng cấp**
- Gửi dữ liệu vào **buffer riêng do lập trình viên cung cấp** bằng `MPI_Buffer_attach`
- Sau khi copy xong vào buffer → hàm trả về
- **Không cần chờ tiến trình nhận sẵn sàng**

3. `MPI_Ssend(...)` (Synchronous Send)

- **Đồng bộ thực sự**
- Gửi thông điệp và **chỉ kết thúc khi bên nhận thực sự đã gọi `MPI_Recv`**
- Đảm bảo rằng **bên nhận đã tiếp nhận trước khi tiếp tục**
- Dùng khi cần **đồng bộ chặt chẽ** giữa các tiến trình

4. `MPI_Irecv(...)` (Immediate/Non-blocking Receive)

- **Không đồng bộ (Non-blocking)**
- Bắt đầu nhận dữ liệu, nhưng **không chờ hoàn thành**
- Tiến trình có thể **tiếp tục công việc khác** và kiểm tra sau bằng `MPI_Wait` hoặc `MPI_Test`
- Rất hữu ích để **tối ưu hiệu năng** (che giấu độ trễ truyền thông)

Câu 5 :

a) Truyền thông bền bỉ (Persistent Messaging) và nhu cầu mô hình hàng đợi thông điệp

Truyền thông bền bỉ (Persistent Messaging) là gì?

Là **cơ chế truyền thông điệp** trong đó **thông điệp vẫn được lưu giữ an toàn** trong hàng đợi (queue) **ngay cả khi hệ thống gặp sự cố, lỗi mạng hoặc quá trình bị ngắt**.

Nói cách khác: Thông điệp **không bị mất** nếu bên nhận chưa kịp xử lý hoặc tạm thời không sẵn sàng.

Tại sao cần mô hình hàng đợi thông điệp (Message Queue)?

- **Tách biệt sender và receiver**: không cần phải online đồng thời.
- **Tăng độ tin cậy**: thông điệp được lưu trữ tạm thời cho đến khi xử lý xong.
- **Chống mất dữ liệu**: thông điệp được ghi xuống ổ đĩa hoặc lưu trữ tạm thời.
- **Giảm độ phụ thuộc thời gian thực**: hỗ trợ mô hình giao tiếp không đồng bộ (asynchronous).

b) Luồng xử lý trong hệ thống Message Queue

Luồng xử lý gồm 3 thao tác chính:

Bước	Mô tả
Put	Tiến trình gửi (sender) **đưa thông điệp** vào hàng đợi
Get / Poll	Tiến trình nhận (receiver) **chủ động kiểm tra (poll)** hoặc lấy thông điệp ra khỏi hàng đợi
Notify	Hệ thống **tự động thông báo** cho receiver khi có thông điệp mới (thay vì receiver phải kiểm tra liên tục)

4 trường hợp hoạt động giữa sender – queue – receiver:

Trường hợp	Mô tả
1. Sender active – Receiver active	Giao tiếp gần như trực tiếp qua hàng đợi (realtime)
2. Sender active – Receiver inactive	Message được giữ trong hàng đợi đến khi receiver sẵn sàng
3. Sender inactive – Receiver active	Receiver chờ sẵn, nhưng không có thông điệp → chờ hoặc bị timeout
4. Sender inactive – Receiver inactive	Thông điệp được lưu trữ lâu dài trong queue (nếu là persistent)

Ví dụ về Message Queue thực tế:

- **RabbitMQ**, **Apache Kafka**, **Amazon SQS**, **ActiveMQ** đều là những hệ thống MQ phổ biến hỗ trợ:
    - Tính bền bỉ (durable queue)
    - Giao tiếp không đồng bộ
    - Load balancing giữa nhiều tiến trình nhận