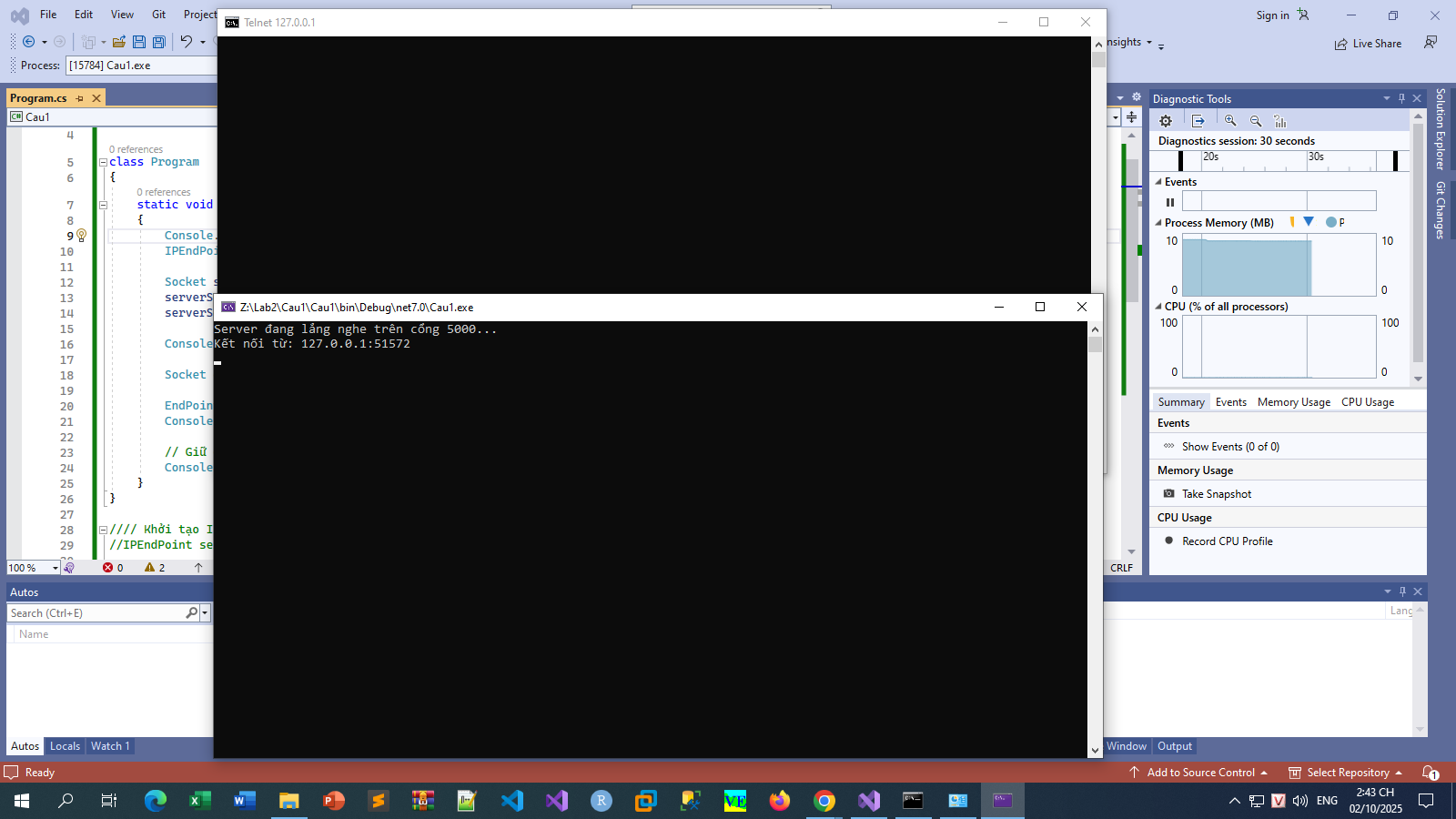
Câu 1:  


Câu 1.3:

1) Khi telnet vào Server bằng port 5000 tại sao port client dùng để kết nối vào lại không phải là port 5000?

 Khi client kết nối tới server, client sẽ sử dụng một **cổng nguồn (source port)** ngẫu nhiên, còn server thì lắng nghe trên cổng đích (destination port) cố định là 5000.

 Cổng nguồn của client được hệ điều hành cấp phát tự động từ khoảng cổng động (dynamic ports), thường từ 1024 đến 65535.

 Việc này nhằm tránh xung đột cổng khi có nhiều kết nối cùng lúc và đảm bảo mỗi kết nối TCP có một cặp địa chỉ IP + port duy nhất.

2) Như hình trên, giải thích vì sao có client dùng port giá trị 1699?

 Cổng 1699 là cổng nguồn do hệ điều hành cấp cho client khi kết nối tới server.

 Đây là một cổng ngẫu nhiên trong khoảng cổng động (thường từ 1024 trở lên), dùng để phân biệt các kết nối khác nhau của client.

 Server sẽ nhận kết nối từ IP client + port 1699 và kết nối này khác biệt so với các kết nối khác.

3) Có phải lúc nào client cũng mở port 1699 để kết nối với Server không?

 Không, **cổng 1699 không phải là cổng cố định của client**.

 Cổng client được hệ điều hành tự động cấp phát và có thể thay đổi mỗi lần kết nối mới.

 Mỗi lần bạn kết nối mới (hoặc khởi động lại client), hệ điều hành sẽ cấp một cổng nguồn khác, không nhất thiết phải là 1699.

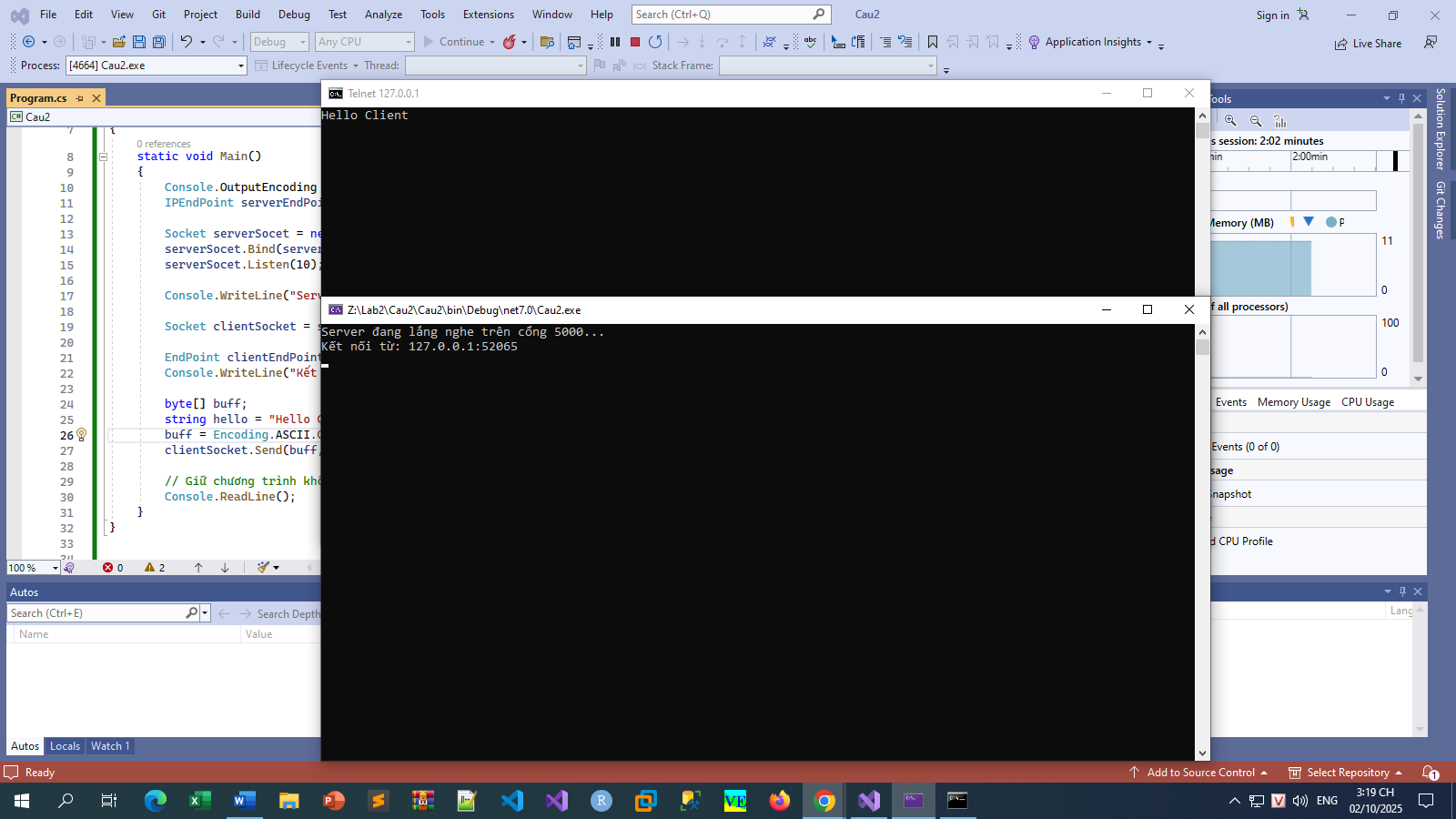
4) Ta có thể chạy 2 chương trình CMD để telnet vào Server trên được không? Giải thích.

 Có thể, bạn hoàn toàn có thể mở nhiều cửa sổ CMD và dùng telnet nhiều lần kết nối tới server trên cùng cổng 5000.

 Server TCP được thiết kế để chấp nhận nhiều kết nối đồng thời, mỗi kết nối sẽ có cặp địa chỉ IP + port nguồn của client riêng biệt.

 server.Listen(10) trong code nghĩa là server có thể để hàng đợi tối đa 10 kết nối chưa xử lý, đủ để chấp nhận nhiều kết nối đồng thời.

 Mỗi kết nối sẽ được server xử lý riêng biệt dựa trên socket client riêng (kết quả trả về từ Accept()).

Câu 2:  


Câu 2.3

1) Tại sao trong đoạn code viết thêm vào ta không cần phải khởi tạo độ lớn ban đầu cho buffer?

* buff được tạo trực tiếp bằng phương thức Encoding.ASCII.GetBytes(hello) — phương thức này **tự động tạo mảng byte đủ lớn** để chứa dữ liệu chuỗi hello

 Nên bạn **không cần phải tạo trước một mảng byte với kích thước cố định** rồi mới đi chuyển dữ liệu vào đó, vì phương thức GetBytes đã xử lý việc cấp phát bộ nhớ cho mảng byte dựa trên độ dài chuỗi.

 Điều này giúp code gọn hơn, tránh phải đoán kích thước buffer và tăng tính linh hoạt.

2) Giải thích ý nghĩa các tham số của phương thức clientSocket.Send

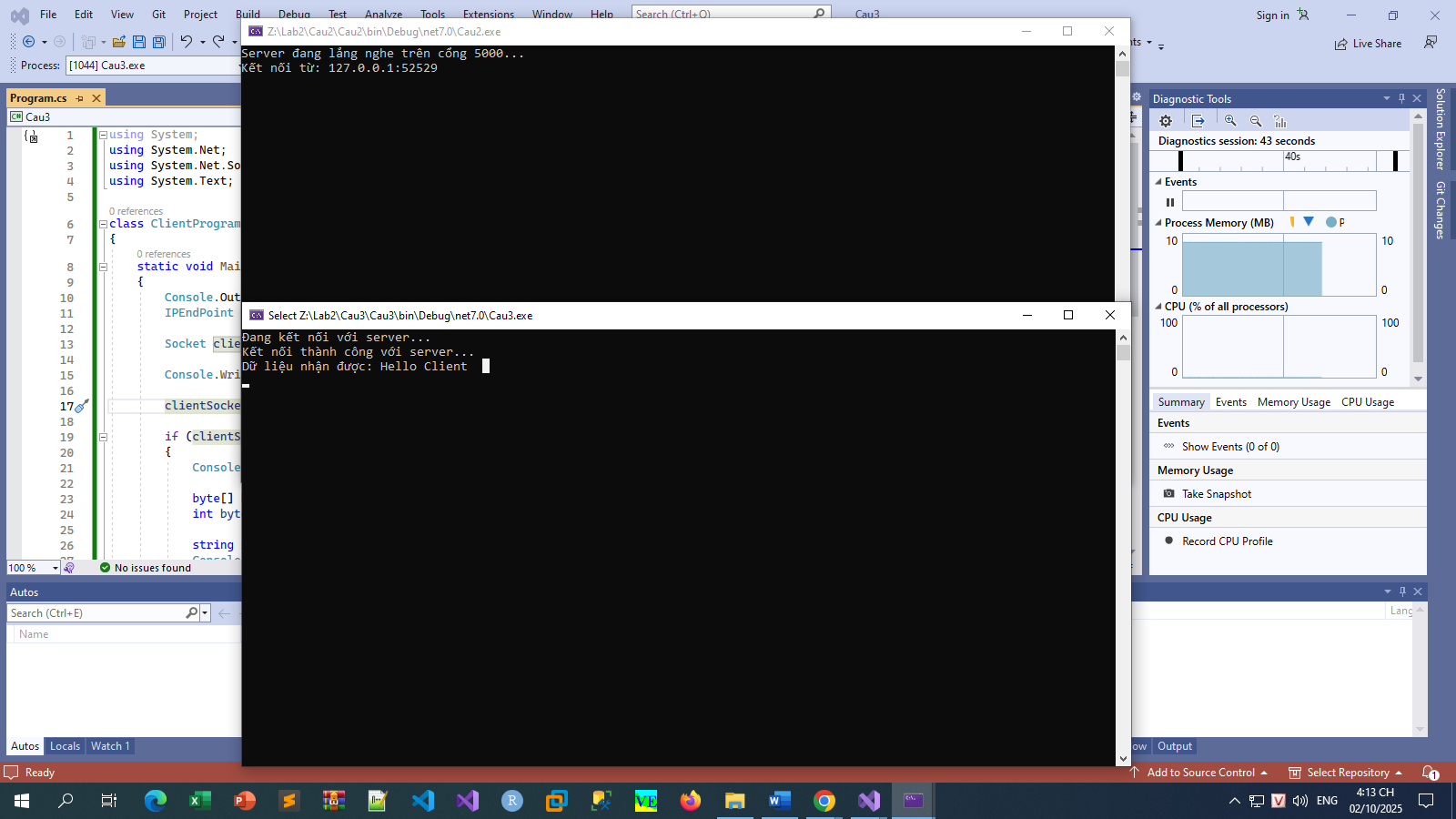
 buff: Mảng byte chứa dữ liệu cần gửi xuống client.

 0: Vị trí bắt đầu trong mảng buff để gửi (offset), ở đây là bắt đầu từ byte đầu tiên.

 buff.Length: Số lượng byte muốn gửi, tức toàn bộ chiều dài của mảng buff.

 SocketFlags.None: Cờ điều khiển hành vi gửi, None nghĩa là không có cờ đặc biệt, gửi dữ liệu bình thường.

Câu 3:



Câu 3.3:  
**1) Khi client kết nối đến server mà server chưa chạy hoặc là đường truyền mạng bị rớt thì có hiện tượng gì xảy ra? Giải thích.**

* **Hiện tượng khi server chưa chạy:**  
  Khi client cố gắng kết nối đến server mà server chưa khởi động hoặc không lắng nghe tại địa chỉ và cổng đó, kết nối sẽ không thành công. Thường thì client sẽ nhận được lỗi kết nối (ví dụ như ConnectionRefusedException trong nhiều ngôn ngữ lập trình). Điều này xảy ra vì không có server nào phản hồi lại yêu cầu kết nối, nên client không thể thiết lập kênh giao tiếp.
* **Hiện tượng khi đường truyền mạng bị rớt:**  
  Nếu đường truyền mạng bị mất kết nối (ví dụ: mất mạng internet, mất kết nối giữa client và server), client sẽ không thể gửi hoặc nhận dữ liệu từ server. Tùy vào giao thức sử dụng, client có thể gặp lỗi timeout khi không nhận được phản hồi trong một khoảng thời gian nhất định, hoặc lỗi mất kết nối đột ngột. Kết nối có thể bị ngắt hoặc không thể thiết lập ban đầu nếu mạng không ổn định.

2) Từ đoạn code ở chương trình client: string str = Encoding.ASCII.GetString(buff, 0, byteReceive);

* Nếu thay byteReceive bằng buff.Length thì chương trình sẽ dịch toàn bộ mảng buff thành chuỗi, kể cả phần chưa nhận dữ liệu (các byte rỗng 0x00). Kết quả là chuỗi thu được sẽ dài hơn, chứa thêm nhiều ký tự rác hoặc khoảng trắng ở cuối, không đúng với dữ liệu thực nhận.

Câu 4:

A screenshot of a computer

AI-generated content may be incorrect.

Câu 5,6:  
A screenshot of a computer

AI-generated content may be incorrect.

Câu 6.3:  
**1) Khi thay đổi kích thước bộ đệm từ 1024 xuống 10 thì có hiện tượng gì xảy ra? Giải thích**

* Khi bộ đệm nhỏ (10 byte), nếu dữ liệu truyền từ client lớn hơn 10 byte thì **không thể nhận hết một lần**.
* Lúc này, socket sẽ phải gọi Receive nhiều lần để nhận đủ toàn bộ dữ liệu.
* Do đó, chuỗi hiển thị có thể bị **cắt khúc** hoặc không đầy đủ nếu lập trình viên không ghép (append) các phần nhận lại.
* Với bộ đệm 1024 byte, thông thường dữ liệu ngắn (ví dụ vài ký tự, câu ngắn) sẽ nhận trọn vẹn một lần, không bị chia cắt.

**2) Nếu ta không reset lại bộ đệm thì có hiện tượng gì xảy ra? Giải thích**

* Bộ đệm là mảng byte dùng chung cho nhiều lần nhận dữ liệu.
* Nếu không reset (khởi tạo lại buff = new byte[1024]; hoặc ghi đè hoàn toàn bằng dữ liệu mới), thì những **byte cũ còn sót lại** trong mảng sẽ vẫn tồn tại.
* Khi dữ liệu nhận mới ngắn hơn dữ liệu cũ, phần dư của dữ liệu cũ vẫn còn → kết quả giải mã ra chuỗi sẽ bị **dư ký tự rác** ở cuối.

Bài tập:  
1)

A screenshot of a computer

AI-generated content may be incorrect.

2)  
A screenshot of a computer program

AI-generated content may be incorrect.

3)  
A screenshot of a computer

AI-generated content may be incorrect.