# CHƯƠNG 3: Luồng và tiến trình

**Họ tên sinh viên: Phan Thành Đạt** **MSSV:20173001**

**Mã Lớp: 118636** **Mã học phần: IT4611**

**Câu 1:**

Cần giới hạn vì mỗi khi khởi tạo một luồng mới sẽ tốn tài nguyên, đến một lúc nào đó sẽ vượt qua giới hạn tài nguyên của hệ thống và khiến hệ thống server bị quá tải. Hơn nữa, việc giới hạn luồng sẽ tạo ra sự ổn định cho server.

**Câu 2:**

Mỗi tiến trình nhẹ nên gắn duy nhất với một luồng đơn để giải quyết bài toán blocking system call khi một luồng nào đó gọi, các luồng khác không bị dừng, không bị ảnh hưởng. Còn nếu gắn với nhiều luồng thì sẽ bị block systemcall. Tiến trình nhẹ là các tiến trình đặc biệt, tức là vẫn được CPU lên lịch nhưng lại có đặc điểm của luồng là có thể trao đổi với các tiến trình nhẹ khác trong cùng một luồng.

**Câu 3:**

Không nên, vì nếu như vậy ta thà gắn thẳng luồng vào tiến trình luôn, không cần tạo ra tiến trình nhẹ, việc ra đời của tiến trình nhẹ là vừa kế thừa được ưu điểm của đa luồng, lại vừa giải quyết được nhược điểm của đa luồng ( một luồng khi gọi Block Systemcall sẽ block các luồng khác cùng tiến trình). Nếu như ta gắn một tiến trình nhẹ vào một tiến trình rồi tạo ra đa luồng trên tiến trình nhẹ đấy thì việc block systemcall trên một luồng cũng làm tiến trình nhẹ đó block các luồng còn lại.

**Câu 4:**

**Xét đơn luồng:**

Thời gian trung bình để nhận và xử lý một yêu cầu:

15*2/3+(15+75)*1/3= 40 (ms)

Số yêu cầu có thể nhận trên 1 giây là :

1000/40 = 25 (yêu cầu/giây)

**Xét đa luồng:**

Do sau khi nhận request, quá trình đọc file được một luồng khác xử lý nên luồng nhận request có thể nhận request khác được luôn

⇨ Số yêu cầu có thể nhận trên 1 giây là:

1000/15 (Yêu cầu/ giây)

**Câu 5:**

Không vô lý bởi vì:

Các user-terminal thực hiện thu nhận các yêu cầu xuất hiển thị đồ họa và hồi âm của người dùng. Phần hiển thị của X cung cấp các dịch vụ hiển thị cho chương trình ứng dụng, và vì vậy nó hành động như một trình phục vụ. Bất cứ một chương trình ứng dụng từ xa nào dùng các dịch vụ này của nó, sẽ hành xử như một trình khách.

**Câu 6:**

Nén thông tin.

**Câu 7:**

| **Đa Luồng** | **Đa Tiến Trình** |
|---|---|
| • Xử lý song song nhiều công việc một lúc | |
| • Chi phí lập trình cao<br>• Chuyển ngữ cảnh tốn ít tài nguyên<br>• Vấn đề Blocked System Call<br>• N luồng chạy nhanh hơn | • Chi phí lập trình thấp<br>• Chuyển ngữ cảnh tốn nhiều tài nguyên<br>• Không gặp vấn đề Blocked System Call<br>• N tiến trình chạy chậm hơn |

**Câu 8:**

Server này là server không trạng thái vì nếu không lưu địa chỉ IP, tức là client kết nối đến như là 1 client mới, server vẫn có thể phục vụ như một client mới. Suy ra là nếu không lưu địa chỉ IP thì server vẫn hoạt động và phục vụ bình thường, tức là server không trạng thái.

**Câu 9:**

Điểm giống: Đều là ảo hóa, giả các interface.

Điểm khác:

| Docker | VM |
|---|---|
| • Các Container tạo ra dùng chung tài nguyên, cần bao nhiêu dùng bấy nhiêu<br>• Tiết kiệm tài nguyên<br>• Do dùng chung OS nên nếu có lỗ hổng nào đấy của host OS thì sẽ ảnh hưởng đến toàn bộ container<br>• Các Container được sử dụng tài nguyên thật, hệ điều hành có thể quy định mức độ tài nguyên khác nhau sao cho vừa đủ tài nguyên để container đó hoạt động.<br>• Cách ly giữa các process | • Mỗi máy ảo sử dụng một OS riêng, được cấp phát bộ nhớ ngay từ đầu tạo ra.<br>• Có máy ảo dư thừa tài nguyên, có máy ảo thiếu tài nguyên => không tối ưu về mặt tài nguyên<br>• Ngược lại, tính an toàn bảo mật tốt hơn<br>• Các OS đều bị giới hạn về tài nguyên<br>• Cách ly giữa các hệ điều hành |