BUỔI 1: TỔNG QUAN VỀ LẬP TRÌNH HỆ THỐNG

(Tutorial / Lab)

HỌ VÀ TÊN (FULL NAME): NGUYỄN TRỌNG HƯỞNG

MÃ SỐ SINH VIÊN (STUDENT ID): 31231023691

LỚP (CLASS): CS0001

MÔN HỌC (SUBJECTS): Lập trình hệ thống – Systems Programming

CHUYÊN NGÀNH (MINOR): Khoa học máy tính – Computer Science

Câu 1 (Nhận biết – Lý thuyết)

Hãy trình bày khái niệm lập trình hệ thống và nêu vai trò của lập trình hệ thống trong phát triển phần mềm. Theo bạn, vì sao lập trình hệ thống lại quan trọng đối với phát triển ứng dụng?

Trả lời câu 1:

1. Khái niệm lập trình hệ thống

- Lập trình hệ thống là quá trình xây dựng các phần mềm đóng vai trò trung gian giữa phần cứng máy tính và phần mềm ứng dụng.
- Khác với lập trình ứng dụng chuyên tạo ra phần mềm cho người dùng cuối như Web, App mobile,… thì mục tiêu của lập trình hệ thống là tạo ra môi trường để các phần mềm khác có thể chạy trên đó.
- Đặc điểm cốt lõi của lập trình hệ thống:
  + Tương tác trực tiếp hoặc rất gần với phần cứng như CPU, Memory, I/O devices.
  + Yêu cầu tối ưu hóa tài nguyên khắt khe như tốc độ xử lý, dung lượng bộ nhớ.
  + Thường sử dụng ngôn ngữ bậc thấp như C, C++, Assembly hay mới đây là Rust.

2. Vai trò của lập trình hệ thống trong phát triển phần mềm

- Resource Management: Hệ điều hành là một sản phẩm của lập trình hệ thống, chịu trách nhiệm phân phối CPU, RAM và thiết bị ngoại vi cho các ứng dụng. Nếu không có lớp này, mỗi ứng dụng sẽ phải tự viết code để điều khiển ổ cứng hay màn hình, điều này là bất khả thi.
- Hardware Abstraction: Nó cung cấp các giao diện lập trình (API) chuẩn. Ví dụ như khi viết Java hay Python, ta chỉ cần dùng lệnh print mà không cần biết máy in hoạt động ra sao. Lập trình hệ thống đã ẩn giấu sự phức tạp của phần cứng đi.
- Xây dựng công cụ phát triển: Các trình biên dịch (Compilers), trình thông dịch (Interpreters), trình gỡ lỗi (Debuggers) và Hệ quản trị cơ sở dữ liệu (DBMS) đều là sản phẩm của lập trình hệ thống.

3. Tầm quan trọng của lập trình hệ thống đối với phát triển ứng dụng

- Performance Optimization:
  + Khi ứng dụng bị chậm, một người hiểu về hệ thống sẽ biết cách nhìn vào cách quản lý bộ nhớ (Stack vs Heap), cách CPU xử lý Multi-threading hay I/O blocking để tối ưu, thay vì chỉ mò mẫm trong code nghiệp vụ.
- Deep Debugging:
  + Có những lỗi như Memory Leak hay Segmentation Fault rất khó hiểu nếu chỉ biết về ngôn ngữ bậc cao. Hiểu về hệ thống giúp ta biết tại sao lỗi đó xảy ra ở tầng thấp để khắc phục triệt để.
- Hiểu về bảo mật:
  + Rất nhiều lỗ hổng bảo mật như Buffer Overflow xuất phát từ việc quản lý bộ nhớ kém. Hiểu về lập trình hệ thống giúp viết code ứng dụng an toàn hơn.
- Khả năng thích nghi công nghệ mới:
  + Các công nghệ hiện đại như Cloud Computing, Containerization (Docker, Kubernetes) hay IoT đều dựa trên các nguyên lý của hệ thống (Linux Kernel, Virtualization). Hiểu gốc rễ giúp ta học các công nghệ này nhanh hơn.

Câu 2 (So sánh – Hiểu)

So sánh lập trình hệ thống và lập trình ứng dụng theo các tiêu chí:

1. Mức độ tương tác với hệ điều hành
2. Quản lý tài nguyên, dữ liệu
3. Yêu cầu về hiệu năng và độ an toàn
4. Nêu ví dụ minh họa cho mỗi loại.

Trả lời câu 2:

Tiêu chí	Lập trình hệ thống	Lập trình ứng dụng
Mức độ tương tác với hệ điều hành	*Tương tác trực tiếp* - Làm việc ở chế độ đặc quyền (Kernel Mode). - Code thường xuyên sử dụng System Calls để giao tiếp thẳng với nhân (Kernel) và phần cứng.	*Tương tác gián tiếp* - Làm việc ở chế độ người dùng (User Mode). - Tương tác với OS thông qua các thư viện, API hoặc Framework đã được trừu tượng hóa. Ví dụ như dùng open() trong Python thay vì xử lý ngắt ổ cứng.
Quản lý tài nguyên, dữ liệu	*Người cấp phát và quản lý* - Chịu trách nhiệm phân phối CPU, RAM, I/O cho các tiến trình khác. - Phải tự quản lý bộ nhớ thủ công (malloc/free) để tránh rò rỉ. Dữ liệu xử lý thường là bit, byte, thanh ghi (registers).	*Người sử dụng* - Sử dụng tài nguyên do hệ thống cấp phát. - Việc quản lý bộ nhớ thường được tự động hóa (Garbage Collection trong Java/Python). - Dữ liệu xử lý là thông tin nghiệp vụ (user data, images, text).
Yêu cầu về hiệu năng và độ an toàn	*Cực kỳ khắt khe* - Hiệu năng: Phải tối ưu từng chu kỳ CPU vì nó là nền tảng cho mọi thứ khác chạy. - An toàn: Lỗi ở tầng này như Kernel Panic, Blue Screen,… có thể làm sập toàn bộ máy tính.	*Linh hoạt hơn* - Hiệu năng: Chấp nhận độ trễ nhất định để đổi lấy tốc độ phát triển (Time-to-market). - An toàn: Lỗi thường chỉ làm tắt ứng dụng đó (Crash App), hệ điều hành vẫn hoạt động bình thường.

Ví dụ minh họa	- Device Drivers: Driver card đồ họa NVIDIA, driver máy in,... - Compilers: GCC, Clang,... - Hệ điều hành: Các thành phần lõi của Linux, Windows Kernel. - Server cấp thấp: Nginx, Redis,…	- Web/Mobile: Facebook, Shopee, Gmail,… - Phần mềm văn phòng: Microsoft Word, Excel,… - Game: League of Legends, PUBG,... - Công cụ: Photoshop, VS Code,..

Tuy nhiên, hiện nay ranh giới này đang mờ dần vì công nghệ ngày càng phát triển. Ví dụ như các trình duyệt web hiện đại như Chrome hay Microsoft Edge quản lý tài nguyên phức tạp không kém gì một hệ điều hành nhỏ, hay ngôn ngữ Rust đang cho phép viết ứng dụng an toàn với hiệu năng của hệ thống.

Câu 3 (Phân tích – Hiểu)

Giải thích mối quan hệ giữa hệ điều hành và chương trình ứng dụng.

Theo bạn, tại sao ứng dụng không thể làm việc trực tiếp với phần cứng mà phải thông qua hệ điều hành?

Trả lời câu 3:

1. Mối quan hệ giữa Hệ điều hành (OS) và Chương trình ứng dụng

- Mối quan hệ này là quan hệ Host - Guest hoặc Nền tảng - Thực thi.
  + Cung cấp môi trường thực thi: Ứng dụng không thể tự chạy lơ lửng trong không gian mà nó cần OS nạp vào bộ nhớ (Loading), cấp phát tài nguyên (CPU, RAM) và quản lý vòng đời (Process Lifecycle).
  + Cung cấp dịch vụ (Services): OS cung cấp một tập hợp các giao diện lập trình gọi là System Calls. Khi ứng dụng cần làm gì đó nặng nhọc như ghi file hay gửi gói tin mạng, nó phải nhờ OS làm hộ thông qua các API này.
  + Kiểm soát và Giới hạn: OS đóng vai trò giám sát ứng dụng để đảm bảo ứng dụng không chiếm dụng quá nhiều RAM hay cố tình truy cập vào vùng nhớ cấm.

2. Tại sao ứng dụng không thể làm việc trực tiếp với phần cứng?

Có 3 lý do cốt lõi khiến ứng dụng không thể làm việc trực tiếp với phần cứng là tam giác An toàn - Tiện lợi - Chia sẻ:

a) Bảo vệ và An toàn (Protection & Security)

- Nếu cho phép ứng dụng chọc trực tiếp vào phần cứng, ví dụ như ghi thẳng vào ổ cứng:
  + Rủi ro sập hệ thống: Một lỗi nhỏ trong code cũng có thể ghi đè lên dữ liệu quan trọng của hệ điều hành, làm hỏng file boot hoặc gây màn hình xanh (BSOD) ngay lập tức.
  + Bảo mật: Một phần mềm độc hại như virus có thể đọc trộm dữ liệu từ RAM của phần mềm ngân hàng hoặc tự ý bật webcam mà không ai kiểm soát được.
  + Cơ chế Protection Rings: CPU hiện đại chia quyền hạn thành các vòng. Ứng dụng chạy ở Ring 3 (User Mode) bị hạn chế quyền năng, trong khi OS chạy ở Ring 0 (Kernel Mode) nắm toàn quyền. Việc cấm truy cập trực tiếp là để duy trì ranh giới an toàn này.

b) Trừu tượng hóa phần cứng (Hardware Abstraction)

- Phần cứng cực kỳ phức tạp và đa dạng, ví dụ như có hàng ngàn loại ổ cứng (SSD, HDD, NVMe) từ hàng trăm hãng khác nhau, mỗi loại có cách giao tiếp điện tử khác nhau nên nếu không có OS, ta phải viết code riêng cho từng loại ổ cứng để lưu một file text. Điều này là bất khả thi.
- OS cung cấp một HAL (Hardware Abstraction Layer). Ta chỉ cần dùng lệnh writeFile(), OS sẽ tự lo phần còn lại với driver tương ứng.

c) Chia sẻ tài nguyên (Resource Multiplexing)

- Máy tính hiện đại là hệ thống đa nhiệm (Multitasking), nếu một ứng dụng như Game được quyền điều khiển trực tiếp Card màn hình và CPU, nó sẽ giữ tài nguyên đó và không nhường cho ai khác. Khi đó, ta sẽ không thể vừa chơi game vừa nghe nhạc hay nhận thông báo Facebook.
- Vì thế OS đóng vai trò là trọng tài (Scheduler), quyết định ai được dùng CPU trong bao lâu, đảm bảo sự công bằng.

Việc ngăn ứng dụng truy cập trực tiếp phần cứng là sự đánh đổi cần thiết: Chúng ta chấp nhận hy sinh một chút hiệu năng do phải đi đường vòng qua OS để đổi lấy sự Ổn định (Stability), Bảo mật (Security) và Khả năng phát triển phần mềm độc lập phần cứng (Portability).

Câu 4 (Vận dụng – Thực hành ngắn)

Sử dụng C#/.NET, hãy viết một chương trình đơn giản để:

In ra thông tin cơ bản về môi trường hệ thống (phiên bản hệ điều hành, thư mục hiện tại, thời gian hệ thống).

Sinh viên giải thích chương trình đang sử dụng dịch vụ nào của hệ điều hành.

Trả lời câu 4:

Chương trình: https://github.com/trongjhuongwr/SystemsProgramming_Labs.git

1. Lấy phiên bản OS (Environment.OSVersion)

- Dịch vụ OS: System Information Service (Dịch vụ thông tin hệ thống).
- Cơ chế:
  + Hệ điều hành lưu trữ thông tin về chính nó (Kernel version, Build number) trong các cấu trúc dữ liệu của nhân (Kernel structures) hoặc Registry (đối với Windows).
  + Khi ta gọi lệnh này, .NET Runtime (CLR) sẽ thực hiện một System Call, ví dụ trên Windows là gọi API GetVersionEx hoặc RtlGetVersion trong ntdll.dll.
  + OS trả về dữ liệu cấu hình tĩnh đã được nạp khi khởi động máy.

2. Lấy thư mục hiện tại (Environment.CurrentDirectory)

- Dịch vụ OS: Process Management (Quản lý tiến trình) và File System (Hệ thống tệp).
- Cơ chế:
  + Mỗi chương trình khi chạy là một Tiến trình (Process). OS quản lý mỗi tiến trình bằng một cấu trúc dữ liệu gọi là PCB (Process Control Block).
  + Trong PCB có chứa thông tin về ngữ cảnh của tiến trình, bao gồm biến môi trường PWD (Print Working Directory) hoặc handle trỏ đến thư mục hiện hành.

+ Lệnh này thực chất là truy vấn thông tin từ PCB của chính tiến trình đó do OS quản lý.

3. Lấy thời gian (DateTime.Now)

- Dịch vụ OS: Timer/Clock Services (Dịch vụ thời gian thực).
- Cơ chế:
  + Máy tính có một chip đếm thời gian thực (Real-Time Clock) chạy bằng pin CMOS trên mainboard.
  + Khi OS khởi động, nó đọc giờ từ RTC và duy trì một bộ đếm phần mềm (System Timer).
  + Lệnh DateTime.Now sẽ kích hoạt một yêu cầu đọc giá trị từ bộ đếm thời gian của OS, thường được cập nhật qua các ngắt phần cứng (Hardware Interrupts).

4. In ra màn hình (Console.WriteLine)

- Dịch vụ OS: I/O Device Management (Quản lý thiết bị nhập/xuất).
- Cơ chế:
  + Chương trình không thể tự vẽ chữ lên màn hình. Nó phải ghi dữ liệu vào một file đặc biệt gọi là Standard Output (stdout).
  + OS nhận dữ liệu từ stdout, gọi trình điều khiển màn hình (Display Driver) để render các pixel chữ tương ứng lên cửa sổ Console.

Câu 5 (Liên hệ – Thảo luận)

Hãy nêu 3 ví dụ bạn cần sử dụng lập trình hệ thống trong quá trình phát triển ứng dụng hoặc phần mềm. Giải thích vì sao các ứng dụng đó cần làm việc gần với hệ điều hành và quản lý tài nguyên hệ thống.

Trả lời câu 5:

Ví dụ 1: Phát triển Game đồ họa cao (Unity, Unreal,...)

Khi ta chơi các game AAA hay phát triển game 3D, lập trình hệ thống là yếu tố sống còn.

- Các game này cần làm việc gần hệ điều hành và quản lý tài nguyên vì:

+ *Tương tác phần cứng (GPU):* Game cần vẽ hàng triệu điểm ảnh mỗi giây. Lập trình viên phải dùng các thư viện đồ họa cấp thấp như DirectX, Vulkan, OpenGL để ra lệnh trực tiếp cho Card đồ họa (GPU) mà không qua quá nhiều lớp trung gian gây chậm trễ.
+ *Kiểm soát bộ nhớ (Manual Memory Management):* Trong game, nếu để hệ thống tự động dọn rác bộ nhớ (Garbage Collection) như Java hay C# làm việc ngẫu nhiên, game sẽ bị khựng như lag hoặc drop FPS ngay lập tức. Lập trình viên hệ thống phải tự tay cấp phát và thu hồi RAM (Object Pooling) để đảm bảo trải nghiệm mượt mà 60-120 FPS.

Ví dụ 2: Các thư viện Trí tuệ nhân tạo (TensorFlow, PyTorch,...)

Ta thường code AI bằng ngôn ngữ Python (bậc cao), nhưng lõi của các thư viện này hoàn toàn là sản phẩm của lập trình hệ thống (C/C++/CUDA).

- Các thư viện này cần làm việc gần hệ điều hành và quản lý tài nguyên vì:
  + *Tính toán song song (Parallel Computing):* Huấn luyện AI cần thực hiện hàng tỷ phép tính ma trận cùng lúc. Lập trình hệ thống cho phép code khai thác các tập lệnh đặc biệt của CPU (AVX, SIMD) hoặc truy cập trực tiếp vào hàng nghìn nhân CUDA của NVIDIA GPU để tăng tốc độ xử lý lên hàng trăm lần so với code Python thuần.
  + *AI tiêu tốn lượng RAM khổng lồ:* Lập trình hệ thống giúp tối ưu hóa cách dữ liệu di chuyển giữa RAM và VRAM để tránh tình trạng nghẽn cổ chai băng thông.

Ví dụ 3: Ứng dụng Hội thoại thời gian thực (Zoom, Microsoft Teams,...)

Khi ta gọi video call, độ trễ (latency) phải cực thấp và âm thanh/hình ảnh phải đồng bộ.

- Các ứng dụng này cần làm việc gần hệ điều hành và quản lý tài nguyên vì:
  + *Truy cập phần cứng I/O (Input/Output):* Ứng dụng cần chiếm quyền điều khiển Micro và Webcam ở mức độ ưu tiên cao nhất, đồng thời sử dụng các chip mã hóa video bằng phần cứng (Hardware Encoding) có sẵn trên CPU/GPU để nén video mà không làm nóng máy.
  + *Quản lý mạng (Network Sockets):* Lập trình hệ thống can thiệp sâu vào chồng giao thức mạng như TCP hoặc UDP stack của hệ điều hành để xử lý mất gói tin (packet loss) và giữ kết nối ổn định ngay cả khi mạng yếu, thay vì phụ thuộc vào cơ chế truyền tin mặc định của trình duyệt web.

Như vậy, lập trình hệ thống không chỉ giới hạn trong việc viết hệ điều hành. Trong phát triển ứng dụng hiện đại, bất cứ khi nào chúng ta chạm đến giới hạn về Hiệu năng (Performance), Thời gian thực (Real-time) hoặc Giới hạn phần cứng (Hardware Constraints), chúng ta buộc phải sử dụng tư duy và kỹ thuật của lập trình hệ thống để giải quyết vấn đề mà lập trình ứng dụng thuần túy không thể làm được.