**BÁO CÁO ĐẦU KỲ**

**ĐỒ ÁN TỐT NGHIỆP – CAPSTONE PROJECT**

(Hợp tác giữa Khoa Điện tử Viễn thông, Trường Đại học Bách khoa – Đại học Đà Nẵng

và

Công ty FPT Software Đà Nẵng)

**Đề tài:** Phần mềm giám sát và tùy chỉnh tải hệ thống IVI

**Thực hiện:** - Hồ Đức Vũ – MSSV: 106200284 – Lớp: 20KTMT2

- Nguyễn Minh Phương – MSSV: 106200241 – Lớp: 20KTMT1

**Hướng dẫn:** TS. Ngô Minh Trí

**Hướng dẫn:** KS. Phan Hồng Sang

**Hướng dẫn:** KS. Nguyễn Việt Đức

Đà Nẵng, tháng 3 năm 2025

**1. Tổng quan về đề tài**

**1.1. Giới thiệu**

Hệ thống IVI (In-Vehicle Infotainment) là một nền tảng công nghệ tích hợp trong ô tô, kết hợp các chức năng giải trí, kết nối và hỗ trợ lái xe. Hệ thống IVI ngày càng trở thành một phần quan trọng trong ngành công nghiệp ô tô hiện đại, giúp nâng cao trải nghiệm người dùng bằng cách tích hợp các dịch vụ như điều hướng GPS, phát nhạc, video, kết nối mạng và hỗ trợ lái xe an toàn. Tuy nhiên, việc quản lý tài nguyên và giám sát hiệu suất hệ thống IVI là một thách thức lớn đối với các nhà sản xuất ô tô và nhà phát triển phần mềm.

**1.2. Tính cấp thiết**

Hệ thống IVI hỗ trợ nâng cao trải nghiệm lái xe, thông qua các dịch vụ phần mềm như điều hướng GPS, phát hiện và nhận diện biển báo, gợi ý nhắc nhở tài xế trong quá trình lái xe, nhưng đặc điểm của hệ thống IVI là một hệ thống có tài nguyên hạn chế, vì vậy có những trường hợp mà hệ thống phải thực thi nhiều ứng dụng và chức năng cùng lúc dẫn đến quá tải. Một số lỗi khi hệ thống IVI quá tải như hệ thống bị treo, chập chờn, gây ra lỗi vận hành và có thể đưa ra những chỉ dẫn và hoạt động sai, điều này có thể gây nguy hiểm lớn đến an toàn của những người trong xe. Những chức năng phổ biến trong hệ thống IVI có thể hoạt động sai lệnh khi bị quá tải như điều hướng bản đồ sai, báo sai tốc độ di chuyển, báo sai biển báo giao thông,... những lỗi đó có thể gây ra hậu quả khiến tài xế đi sai luật và có thể dẫn đến tai nạn giao thông, tạo ra sự khó chịu khi sử dụng hệ thống. Đôi khi trong các trường hợp hệ thống IVI quá tải nghiêm trọng dẫn đến nhiều chức năng bị dừng đột ngột, điều năng rất nguy hiểm khi đang xe di chuyển ở tốc độ cao.

**1.3. Vấn đề cần giải quyết**

Để hệ thống IVI hoạt động ổn định, cung cấp dịch vụ tốt nhất và an toàn đến người dùng, nhà sản xuất và nhà phát triển phần mềm cần tạo ra các dịch vụ đảm bảo phù hợp với điều kiện tài nguyên của hệ thống, tránh các trường hợp những dịch vụ hoạt động trên hệ thống chiếm một lượng lớn tài nguyên dẫn đến hệ thống bị quá tải. Song hành với đó, ta cần theo dõi tải của hệ thống IVI trong quá trình vận hành thực tế, với mục đích khắc phục kịp thời sự cố khi hệ thống IVI quá tải bị quá tải, đồng thời cung cấp thông tin để đội vận hành hệ thống có thể nhận biết và chuẩn đoán nguyên nhân lỗi một cách chuẩn xác. Vấn đề đặt ra là làm cách nào để nhà sản xuất và nhà phát triển phần mềm có thể điều chỉnh dịch vụ của mình phù hợp với tài nguyên của hệ thống, đồng thời có thể theo dõi hệ thống trong quá trình vận hành?

**2. Giải pháp đặt ra**

Từ vấn đề trên, ta có thể thấy rằng giải pháp được chia làm 2 giai đoạn, giai đoạn thứ nhất là giai đoạn phát triển hệ thống và giai đoạn thứ hai là giai đoạn vận hành hệ thống.

Tại giai đoạn phát triển hệ thống, ta cần kiểm tra và đánh giá lưu lượng tài nguyên mà các dịch vụ, phần mềm (gọi chung là tiến trình) đang được phát triển sử dụng, từ đó có thể đưa ra những nhận định về các lỗi tiềm ẩn trong thiết kế hoặc tương tác giữa các tiến trình, trong môi trường testing, ta có thể dễ dàng thiết lập các test case, đảm bảo phát hiện và khắc phục hầu hết (lý tưởng nhất là triệt để) các lỗi dẫn đến hệ thống quá tải. Hiểu được điều này, chúng tôi đề xuất một giải pháp đó là thiết kế và phát triển phần mềm có thể thu thập dữ liệu tài nguyên trên hệ thống, hiển thị dữ liệu đó theo thời gian thực để mô tả xu hướng, lưu lượng tài nguyên được sử dụng trên hệ thống, từ đó giúp nhà sản xuất và nhà phát triển có thể quan sát được các giá trị tài nguyên đang sử dụng để đánh giá và điều chỉnh các tiến trình sao cho phù hợp với giới hạn phần cứng của hệ thống.

Tại giai đoạn vận hành hệ thống, đây là lúc hệ thống được đưa vào sử dụng thực tế, lúc này ta cần quan sát hệ thống với mục đích bảo trì và khắc phục kịp thời khi sự cố xảy ra. Với mục tiêu là đảm bảo hệ thống hoạt động ổn định, không bị quá tải, phần mềm do chúng tôi đề xuất đồng thời có thể đánh giá cơ bản về mức độ sử dụng tài nguyên của hệ thống theo thời gian thực, từ đó đưa ra cảnh báo và xử lý kịp thời khi hệ thống có dấu hiệu quá tải thông qua việc tùy chỉnh tải trên hệ thống, cùng với đó có thể lưu trữ dữ liệu tài nguyên của hệ thống vào cơ sở dữ liệu giúp đội ngũ kỹ thuật dễ dàng theo dõi xu hướng sử dụng tài nguyên, phát hiện các vấn đề tiềm năng cũng như hiểu rõ nguyên nhân dẫn đến quá tải nếu có xảy ra, từ đó đưa ra các phương án cải thiện hiệu suất hệ thống.

Từ những phân tích trên, chúng tôi đã đưa ra đề xuất về phần mềm có thể đáp ứng cho 2 giai đoạn: phát triển và vận hành. Trong giai đoạn phát triển, việc giám sát và đánh giá mức độ sử dụng tài nguyên trên toàn hệ thống và trên từng tiến trình giúp nhà phát triển tối ưu hóa các phần mềm, giảm thiểu nguy cơ quá tải ngay từ khâu thiết kế. Trong giai đoạn vận hành, phần mềm tiếp tục đóng vai trò quan trọng bằng cách theo dõi tài nguyên hệ thống theo thời gian thực, phát hiện các dấu hiệu quá tải, từ đó có thể xử lý kịp thời và cũng như hỗ trợ cho quá trình bảo trì hệ thống.

**3. Sản phẩm dự kiến đạt được**

Với đề tài: “Phần mềm giám sát và tùy chỉnh tải hệ thống IVI”, chúng tôi dự kiến sẽ triển khai dự án với hai thành phần chính hoạt động trên hai thiết bị khác nhau. Thành phần PerformanceViewer App là ứng dụng Windows (sử dụng Qt/QML với C++) dùng để hiển thị tình trạng hệ thống và điều khiển khi quá tải. Thành phần Performance Service là ứng dụng chạy trên thiết bị Linux (mô phỏng cho hệ thống In-Vehicle Infotainment – IVI) thu thập dữ liệu tài nguyên. Mục tiêu của hệ thống là giám sát tài nguyên (CPU, MEM) cũng như hiệu năng của các tiến trình trên thiết bị IVI, đưa ra cảnh báo khi quá tải và chủ động xử lý (ví dụ kết thúc tiến trình tiêu tốn nhiều tài nguyên) để đảm bảo hệ thống vận hành ổn định. Hai thành phần này giao tiếp với nhau qua giao thức TCP/IP trên mạng nội bộ để trao đổi dữ liệu theo thời gian thực. Giao tiếp TCP/IP đảm bảo việc truyền dữ liệu tin cậy, có thứ tự và được kiểm soát lỗi giữa hai ứng dụng​, nhờ đó số liệu giám sát từ hệ thống IVI (Linux) được gửi đi một cách chính xác và đầy đủ.

**3.1 Thành phần Windows (PerformanceViewer App)**

Ứng dụng giám sát chạy trên Windows App Desktop được phát triển bằng Qt/QML, nhằm cung cấp giao diện trực quan để hiển thị dữ liệu hệ thống thời gian thực nhận được từ hệ thống IVI. Trong hệ thống phần mềm này, PerformanceViewer App (gọi là Windows App) đóng vai trò như một Server, liên tục lắng nghe các kết nối từ Performance Service và xử lý luồng dữ liệu được gửi đến đích. Windows App sẽ biểu diễn dữ liệu bằng các biểu đồ trực quan nhằm thuận tiện cho việc quan sát các thay đổi. Theo như Hình 3.1.1, sẽ có một biểu đồ biểu diễn mức sử dụng của từng lõi CPU (core), mỗi màu tương ứng với mỗi core và một biểu đồ biểu diễn MEM. Đồng thời, Windows App liệt kê danh sách các tiến trình đang chạy trên hệ thống IVI kèm theo mức sử dụng CPU, MEM của từng tiến trình (*Hình 3.1.2*). Khi phát hiện hệ thống quá tải (theo ngưỡng định nghĩa trước: ví dụ CPU > 85% và RAM > 90% duy trì liên tục trong hơn 2 phút), ứng dụng sẽ kích hoạt cảnh báo: có thể hiển thị một Pop-up cảnh báo, và đồng thời gửi tín hiệu đến Performance Service để phát âm thanh cảnh báo (pip-pip-pip) ngay trên hệ thống IVI. Quan trọng hơn, nó sẽ thực hiện một thuật toán quyết định xem tiến trình nào cần được kết thúc để giảm tải: dựa trên độ ưu tiên (priority) của tiến trình và mức tài nguyên mà tiến trình đó sử dụng. Ví dụ, tiến trình có mức ưu tiên thấp nhưng đang chiếm CPU hoặc RAM cao sẽ bị đề xuất dừng trước. Sau khi quyết định, sẽ gửi lệnh qua kết nối TCP/IP yêu cầu Performance Service kết thúc tiến trình tương ứng. Toàn bộ sự kiện và số liệu cũng được ghi lại: định kỳ (mỗi 2 giây) ghi số liệu tổng quan vào file log (Excel hoặc CSV) làm lịch sử, và khi xảy ra quá tải thì ghi lại chi tiết (snapshot danh sách tiến trình, thời điểm quá tải, tiến trình đã bị kết thúc, v.v.).

*Hình 3.1.1: Hiển thị thông tin về tổng CPU, MEM của hệ thống lên DashBoard.*

A screenshot of a computer

AI-generated content may be incorrect.

*Hình 3.1.2: Hiển thị thông tin chi tiết về mức chiếm dụng tài nguyên của từng tiến trình (kết quả dự kiến).*

**3.2 Thành phần Linux (Performance Service)**

Performance Service (còn gọi là Linux App) được phát triển trên môi trường Linux và hoạt động ngay trên thiết bị với nhiệm vụ thu thập dữ liệu tài nguyên của hệ thống IVI bao gồm tài nguyên được sử dụng trên toàn hệ thống và tài nguyên bị chiếm dụng bởi các tiến trình hoạt động trên hệ thống, sau đó gửi những dữ liệu này đến PerformanceViewer App thông qua giao thức TCP/IP.

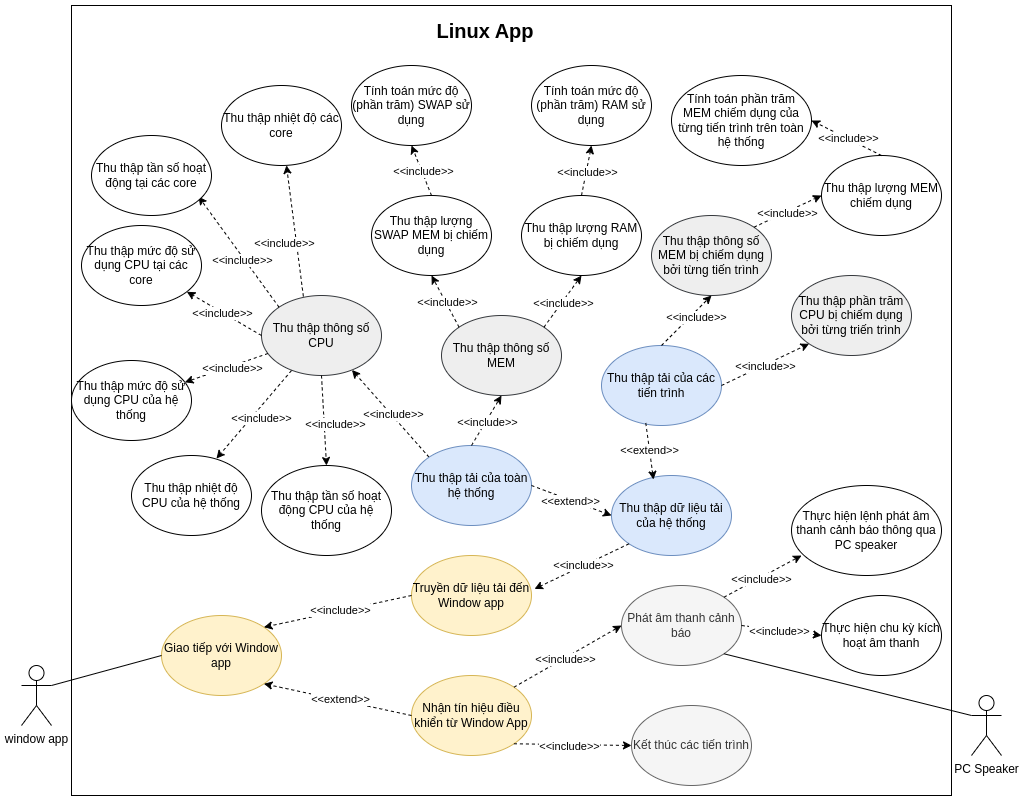
Các tài nguyên được thu thập đối với toàn hệ thống bao gồm CPU, bộ nhớ RAM và bộ nhớ SWAP. Còn đối với tài nguyên trên các tiến trình, ta thu thập phần trăm CPU, MEM mà mỗi tiến trình chiếm dụng. Lý do thu thập các dữ liệu đó là vì khi ta xét hệ thống có quá tải hay không thì tài nguyên ảnh hưởng lớn nhất đó chính là bộ nhớ, đây là tài nguyên được sử dụng nhiều nhất và có nguy cơ bị quá tải nhất, tiếp đến là mức chiếm dụng CPU, bao gồm mức CPU đang sử dụng, nhiệt độ, và tần số, các thông số này ảnh hưởng đến lớn hiệu suất thực thi các tiến trình trên hệ thống, khi các thông số này trên CPU ở mức cao liên tục sẽ dẫn đến hiệu suất hệ thống giảm, hoạt động các tiến trình bị sai lệnh đi và có thể dẫn đến treo hệ thống. Đối với các thông số khác như Network, Disk thì có khả năng thể hiện tốc độ thực thi của các tiến trình nhưng ảnh hưởng rất ít đến việc khiến hệ thống bị quá tải.

Song song với nhiệm vụ thu thập dữ liệu tài nguyên thì Performance Service còn có thể can thiệp vào hệ thống để tiến hành kết thúc các tiến trình chiếm dụng mức tài nguyên lớn theo tín hiệu nhận được từ PerformanceViewer App thông qua giao thức TCP/IP đồng thời phát cảnh báo đến người dùng qua Speaker trên hệ thống để người dùng ý thức được tình hình của hệ thống từ đó đưa ra những nhận định chính xác hơn.

**4. Thiết kế hệ thống**

**4.1. Các chức năng của hệ thống phần mềm**

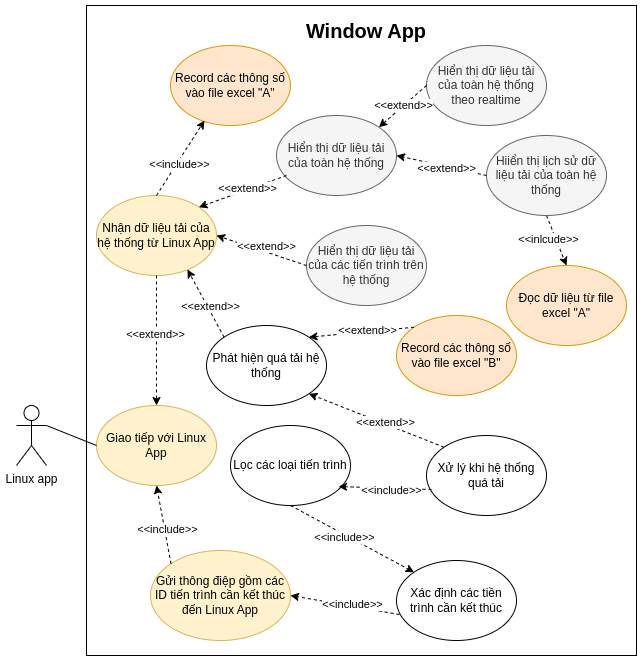
Đối với việc thiết kế các chức năng cho hệ thống, chúng tôi tách riêng thành 2 phần, Một phần thiết kế cho các chức năng thuộc về Performance Service như được mô tả tại *Hình 4.1.1.* Một phần khác thiết kế cho các chức năng thuộc về PerformanceViewer App như mô tả tại *Hình 4.1.2.*

*Hình 4.1.1: Sơ đồ mô tả các chức năng và mối quan hệ của chúng trên Performance Service (Linux App)*

Đầu tiên ta sẽ phân tích thiết kế trên Performance Service (Linux App), với mục tiêu sản phẩm đầu ra như mô tả tại *Mục 3.2,* chúng tôi đã thiết kế Linux App với các chức năng thu thập thông số CPU, bao gồm thu thập mức độ sử dụng CPU của hệ thống, mức độ sử dụng CPU tại từng core, trên thực tế thì mức độ sử dụng CPU của hệ thống là thể hiện cho trung bình tổng của tất cả các core, ngoài mức độ sử dụng CPU, chúng tôi còn lấy thêm thông số về nhiệt độ và tần suất hoạt động của CPU tại các core và cũng như của toàn hệ thống, những chức năng này có mối quan hệ mật thiết với nhau, nên được thiết kế trong cùng một module được gọi là “Thu thập thông số CPU". Đối với dữ liệu tài nguyên của toàn hệ thống thì chúng tôi còn thu thập thêm 2 thông số nữa bao gồm RAM và SWAP MEM, sau khi thu thập thì cả 2 thông số này được tính toán mức phần trăm mà hệ thống đang sử dụng, nhìn chung thì RAM và SWAP MEM đều là các thông số biểu thị cho mức tài nguyên bộ nhớ mà hệ thống sử dụng, nên chúng tôi nhóm 2 thông số này vào cùng một module được gọi là “Thu thập thông số MEM". Cả 2 module “Thu thập thông số CPU" và “Thu thập thông số MEM" mô tả tài nguyên tải của toàn hệ thống, ngoài ra thì Linux App còn thu thập thêm các thông số của từng tiến trình đang hoạt động trên hệ thống, mối tiến trình sẽ được thu thập các thông số bao gồm phần trăm CPU đang chiếm dụng, mức bộ nhớ mà nó đang dùng và đồng thời tính toán phần trăm cho thông số để mô tả trực quan hơn.

Các chức năng ở trên được thiết kế để thu thập dữ liệu tài nguyên trên hệ thống, sau đó thì các dữ liệu này sẽ được gửi đến PerformanceViewer App (Window App), từ tư duy đó, chúng tôi thiết kế một module “Giao tiếp với Window App" với mục đích khởi tạo kết nối giữa 2 ứng dụng (Window App và Linux App), module này có 2 chức năng bao gồm truyền dữ liệu đến Window App và nhận tín hiệu điều khiển từ Window App, đối với truyền dữ liệu thì ta sẽ truyền dữ liệu tài nguyên như đã đề cập, đối với nhận tín hiệu điều khiển thì chức năng này được thiết kế với mục đích tùy chỉnh tải cho hệ thống và phát cảnh báo qua Speaker đến người dùng khi hệ thống quá tải.

Tiếp theo ta sẽ phân tích thiết kế trên PerformanceViewer App (Window App), dựa trên mục tiêu sản phẩm đầu ra được mô tả tại M*ục 3.1,* Window App được thiết kế với các chức năng bao gồm: hiển thị dữ liệu tải của toàn bộ hệ thống, hiển thị dữ liệu tải của các tiến trình trên hệ thống, đánh giá và phát hiện quả tải hệ thống sau đó đưa ra quyết định kết thúc các tiến trình đang hoạt động. Chúng tôi thiết kế mối quan hệ giữa các chức năng như Hình 4.1.2, Window App giao tiếp với Linux App để truyền và nhận tín hiệu, sau đó dữ liệu này sẽ được lưu vào datasheet, cùng với đó ta có thể xem dữ liệu tải toàn hệ thống, hoặc dữ liệu tải trên từng tiến trình, Window App sẽ theo giõi và đánh giá liên tục về trạng thái tải hiện tải, đưa ra kết luận hệ thống có quá tải, nếu có thì ta sẽ xử lý bằng cách lọc các tiến trình người dùng, sau đó xác định các tiền trình cần kết thúc và gửi danh sách này đến Linux App.



*Hình 4.1.2: Sơ đồ mô tả các chức năng và mối quan hệ của chúng trên PerformanceViewer App (Window App)*

**4.2. Trình tự tương tác của các đối tượng trong hệ thống theo thời gian**

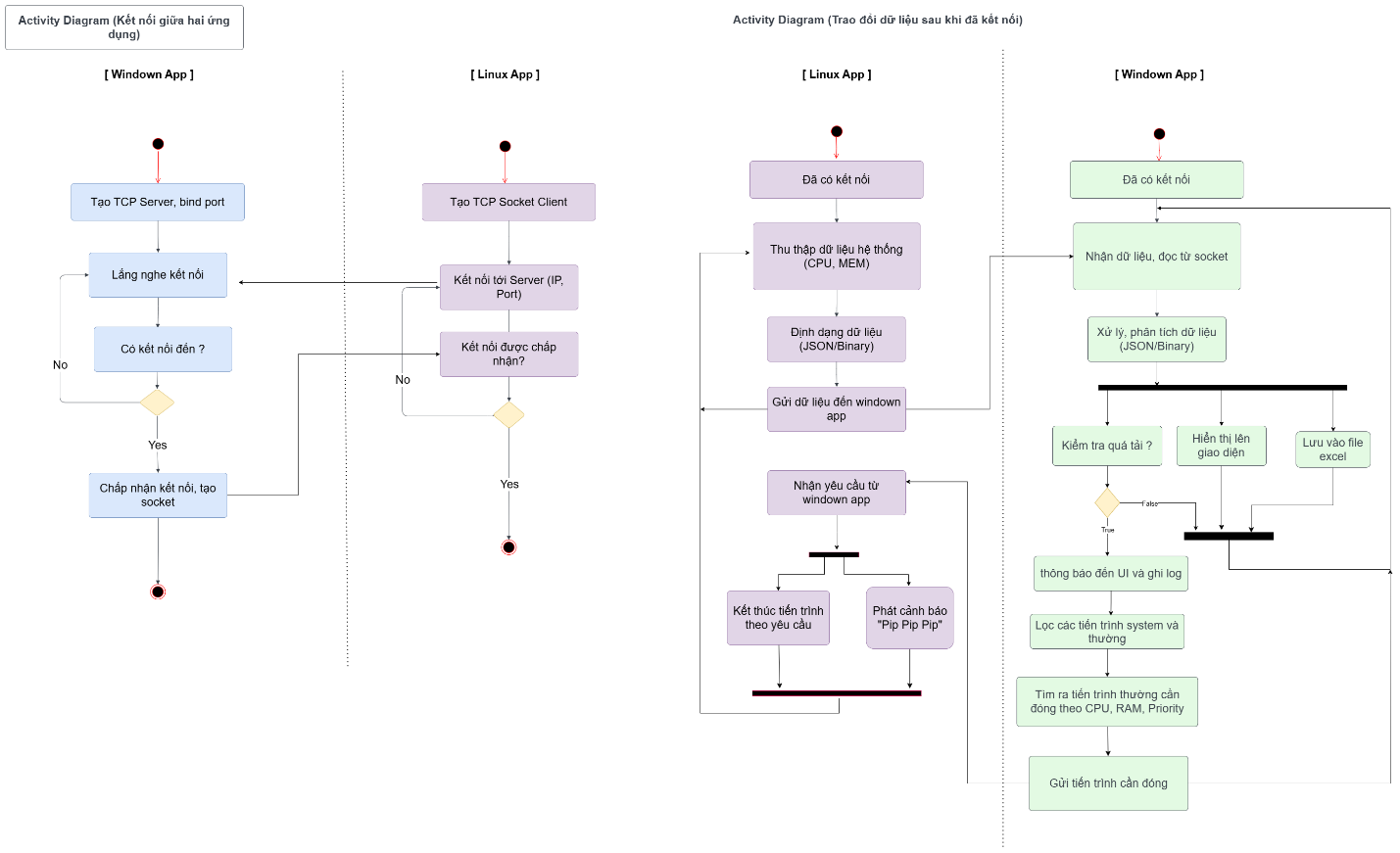
Trong phần mềm này, chúng tôi xây dựng 2 ứng dụng, nhưng khi xét theo hướng các tác nhân và đối tượng hoạt động trong phần mềm, chúng tôi xác định 4 đối tượng bao gồm: thiết bị IVI (hệ thống), Performance Service (Linux App), PerformanceViewer App (Window App) và Excel (Datasheet dùng để lưu trữ dữ liệu của phần mềm) như mô tả tại Hình 4.2.

|  |
| --- |
|  |
| *Hình 4.2: Sơ đồ thể hiện luồng hoạt động theo thời gian của phần mềm* |

Để bắt đầu, Linux App được khởi động trên thiết bị IVI sẽ gửi yêu cầu kết nối TCP/IP đến Window App, sau đó Window App sẽ gửi phản hồi chấp nhận kết nối. Thứ tự thực hiện tiếp theo, Linux App sẽ tương tác với hệ thống để thu thập dữ liệu tải, sau đó dữ liệu này sẽ được gửi từ Linux App đến Window App. Lúc này tại Window App, ta sẽ thực hiện các luồng hoạt động độc lập, bao gồm các chức năng thực hiện độc lập như: lưu trữ dữ liệu vào sheet 1, hiểu thị dữ liệu tải và kiểm tra đánh giá mức tải của hệ thống, đối với thời điểm mà Window App đánh giá hệ thống đang quá tải, ta sẽ thực hiện 2 công việc song song đó là lưu trữ dữ liệu quá tải này vào sheet 2, cùng với đó sẽ tiến hành lần lượt các công việc lọc tiến trình hệ thống và tiến trình thường, kiểm tra tài nguyên và mức ưu tiên của từng tiến trình từ đó xác định được danh sách các tiến trình cần kết thúc và gửi đến Linux App. Sau khi nhận được danh sách các tiến trình từ Window App, Linux App sẽ tiến hành các lệnh kết thúc các tiến trình đó trên hệ thống, đồng thời phát tín hiệu tạo âm cảnh báo đến Speaker trên hệ thống.

**4.3. Luồng thực thi chính của phần mềm**

Sơ đồ hoạt động (Hình 4.3) diễn tả luồng hoạt động chính của hệ thống dưới dạng một quy trình làm việc bao gồm các bước rõ ràng và các hành động cụ thể. Sơ đồ này phân tách hai luồng xử lý song song trên Linux App và Windows App, điều này cho thấy sự tương tác giữa chúng. Sơ đồ này giúp người thiết kế xác định rõ cần những bước xử lý nào và trong thứ tự logic nào.

*Hình 4.3: Luồng thực thi chính của hệ thống*

Trên Linux App, hoạt động bắt đầu khi ứng dụng được khởi chạy trên hệ thống IVI. Khởi tạo một QtcpClient và cố gắng kết nối đến Windows App với địa chỉ IP và Port được cung cấp. Khi kết nối thành công, bắt đầu thu thập dữ liệu hệ thống định kỳ, dữ liệu này bao gồm CPU, MEM của hệ thống và danh sách các tiến trình đang hoạt động cùng với thống tin chi tiết của từng tiến trình. Sau đó dữ liệu được định dạng, đóng gói rồi gửi đến cho Windows App. Quá trình này sẽ được lặp đi lặp lại cho đến khi chương trình được kết thúc.

Trên Windows App, sau khi khởi động, Windows App tạo một QTcpServer và lắng nghe kết nối tại địa chỉ IP và Port được cấu hình trước. Khi nhận tín hiệu kết nối đến, hệ thống sẽ chấp nhận và khởi tạo một QTcpSocket mới để phục vụ cho kết nối này. Để tối ưu hiệu năng, kết nối sẽ được gán cho một luồng riêng biệt (dùng QThread), đảm bảo rằng việc nhận dữ liệu từ Linux App sẽ không ảnh hưởng đến giao diện người dùng (UI). Dữ liệu nhận được sẽ được xử lý thành 2 loại, đó là dữ liệu về tải hệ thống và dữ liệu của các tiến trình. Sau bước này, luồng hoạt động phân làm 3 nhánh, đầu tiên là nhánh kiểm tra tình trạng quá tải, tiếp đến là nhánh hiển thị lên giao diện, cuối cùng là nhánh lưu dữ liệu xuống file excel. Các nhánh này được mô tả là các hoạt động đồng thời. Tại nhánh kiểm tra quá tải, khi xác định hệ thống được cho là quá tải thì hệ thống sẽ phát tín hiệu quá tải để hiển thị thông báo lên giao diện và đồng thời ghi log lại sự kiện đó. Sau đó tiến hành xác định các tiến trình chiếm nhiều tài nguyên, dựa trên CPU, RAM và độ ưu tiên để chọn ra tiến trình nên kết thúc và gửi lệnh điều khiển về cho Linux App để tiến hành đóng tiến trình.

**4.4. Thiết kế đối tượng**

**4.4.1. Thiết kế đối tượng trên Performance Service**

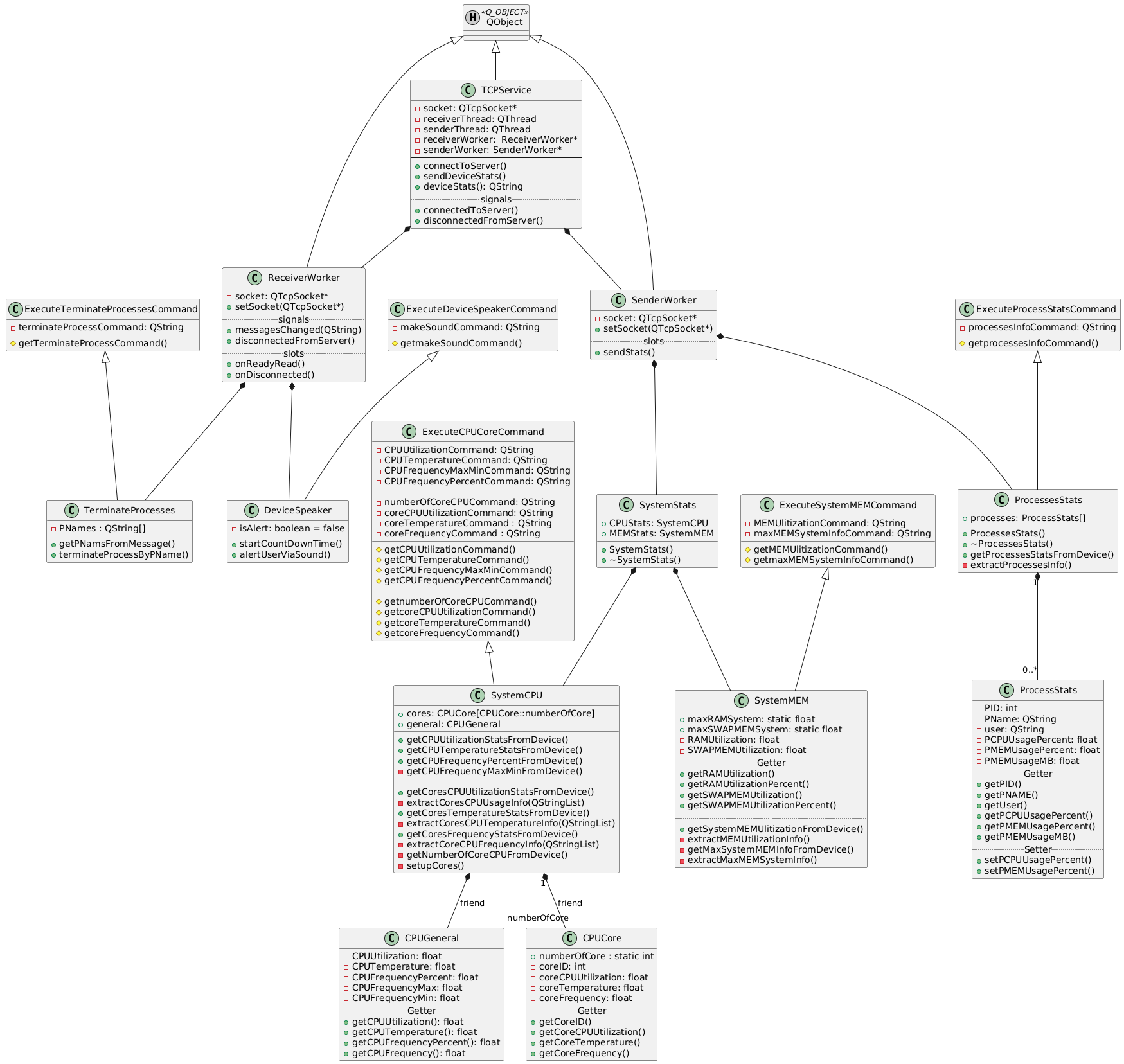
Hình 4.4.1 mô tả cấu trúc của các chức năng được thiết kế theo hướng đối tượng hóa, với việc thiết kế theo OOP giúp phần mềm dễ dàng triển khai, phát triển và bảo trì hơn.

Tại đây, chúng tôi thiết kế Performance Service thành các tầng bao gồm:

1. Tầng giao tiếp mạng.
2. Tầng thực thi lệnh.
3. Tầng quản lý tài nguyên.
4. Tầng thực thi tùy chỉnh tải.

Tầng giao tiếp mạng bao gồm các lớp TCPService, ReceiverWorker và SenderWorder, với thiết kế được chi thành 2 luồng nhận (ReceiverWorker) và gửi (SenderWorder), và 2 luồng này được quản lý trực tiếp bởi TCPService.

Tầng thực thi lệnh được thiết kế với mục đích triển khai các lệnh từ Performance Service đến hệ thống, tầng này bao gồm các câu lệnh được triển khai bên trong các lớp Execute\*Command, và được kế thừa bởi các đối tượng cần triển tương tác với hệ thống, các đối tượng này được thiết kế tại tầng quản lý tài nguyên và tầng thực thi tùy chỉnh tải.



*Hình 4.4.1: Thiết kế đối tượng trên Linux App*

Tầng quản lý tài nguyên là tầng sử dụng các câu lệnh được kế thừa từ tầng thực thi để triển khai thu thập và lưu trữ dữ liệu tài nguyên của hệ thống, được thiết kế theo nguyên tắc code sạch (clear code), mối class được thiết kế tinh chỉnh, dễ triển khai, dễ bảo trì và mở rộng, mỗi class chi thực thi một nhiệm vụ duy nhất. Tầng này bao gồm các lớp SystemStats, và ProcessesStats, ở mức chi tiết hơn chúng tôi thiết kế thêm các đối tượng con thuộc 2 lớp trên. Trong SystemStats, chúng tôi đối tượng hóa những thuộc tính của lớp này thành 2 lớp là CPUGeneral đại diện cho các thông số CPU của toàn hệ thống và CPUCore đại diện cho các thông số của các core CPU. Trong ProcessesStats, chúng tôi đối tượng hóa mỗi process để có thể dễ dàng quản lý và mở rộng chức năng.

Tầng thực thi tùy chỉnh tải có nhiệm vụ xử lý quá tải hệ thống, tại đây có các lớp TerminateProcesses dùng để kết thúc các tiến trình theo danh sách nhận được, DeviceSpeaker dùng để phát âm thanh cảnh báo khi nhận được tín hiệu hệ thống đang quá tải.

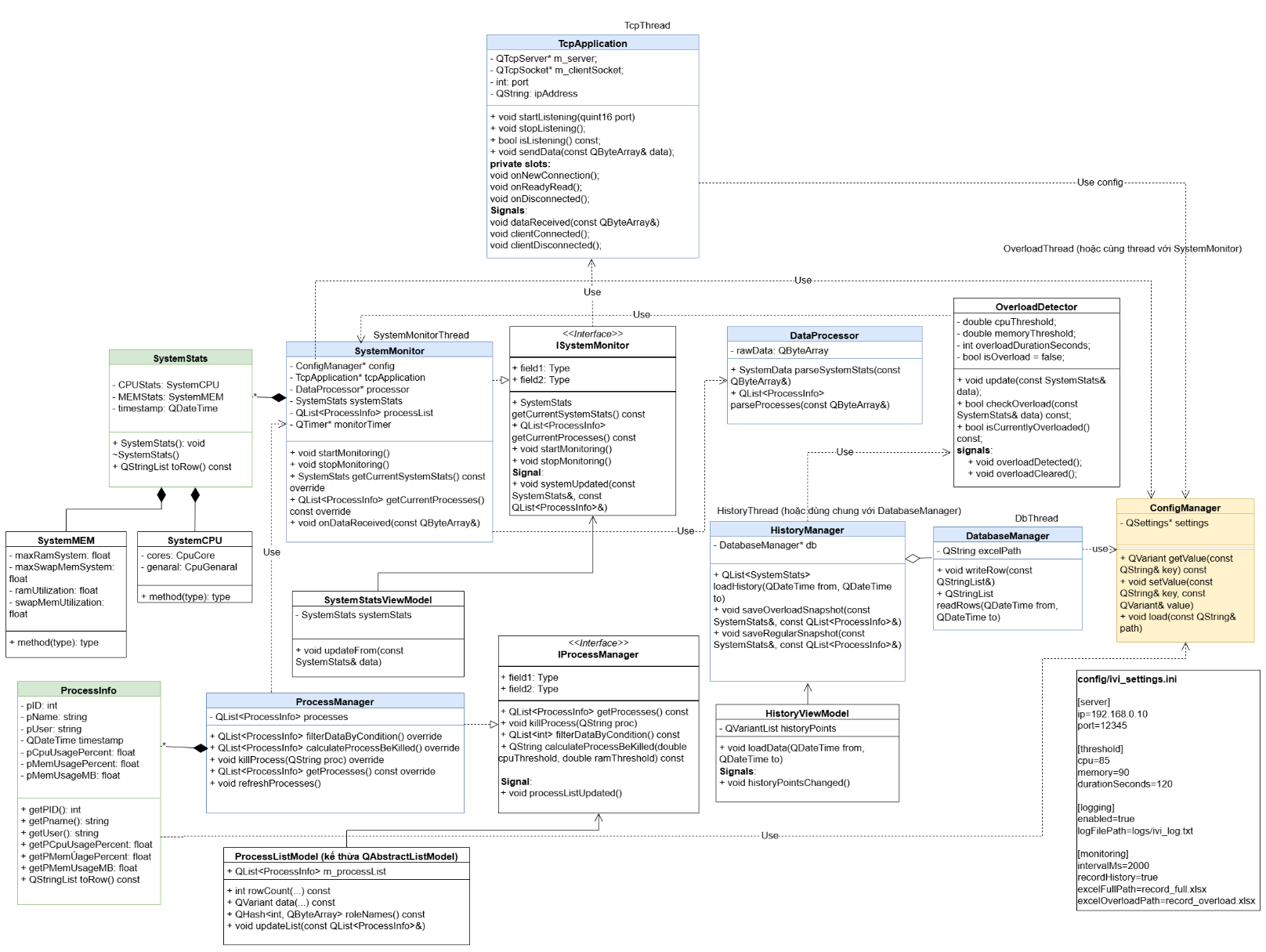
**4.4.2. Thiết kế đối tượng trên PerformanceViewer App**

Sơ đồ lớp (Hình 4.4.2) mô tả cấu trúc chi tiết của toàn bộ hệ thống, bao gồm các lớp chính, thuộc tính và phương thức của từng lớp, cũng như là mối quan hệ giữa các lớp với nhau.

Hệ thống được xây dựng theo mô hình kiến trúc hướng đối tượng (OOP), chia thành nhiều module độc lập, có tính đóng gói và dễ mở rộng. Mô hình lớp được thiết kế theo tiêu chuẩn Clean Architecture (Clean Architecture là một mô hình kiến trúc phần mềm do Robert C. Martin (Uncle Bob) đề xuất, nhằm đảm bảo sự tách biệt rõ ràng giữa các tầng của hệ thống, từ đó giúp ứng dụng dễ mở rộng, dễ kiểm thử và bảo trì lâu dài.) kết hợp với mô hình MVVM (Model-View-ViewModel; Model là dữ liệu và xử lý logic trong hệ thống; View là giao diện người dùng – là các file QML; ViewModel là cầu nối giữa Model và View, cung cấp dữ liệu dưới dạng Q\_PROPERTY cho QML) trong Qt/QML. Việc phân lớp rõ ràng giúp tách biệt các thành phần xử lý, lưu trữ dữ liệu, giao tiếp mạng, giao diện người dùng và các chức năng mở rộng khác, từ đó tăng tính linh hoạt và khả năng tái sử dụng mã nguồn.

Tổng thể, hệ thống được chia thành các nhóm lớp chính:

1. Lớp xử lý giám sát (SystemMonitor, ISystemMonitor): đóng vai trò trung tâm trong việc thu nhận, xử lý dữ liệu từ hệ thống IVI. SystemMonitor kế thừa từ ISystemMonitor (một lớp Interface), có nhiệm vụ nhận dữ liệu từ TcpSocket thông qua TcpServer, sau đó chuyển đến DataProcessor để phân tách và cấu trúc dữ liệu. Sau khi xử lý, SystemMonitor phát tín hiệu systemUpdated(...) để thông báo cho các thành phần giao diện hoặc lưu trữ dữ liệu biết để có thể cập nhật lại giao diện, cũng như lưu trữ.
2. Lớp mô hình dữ liệu (SystemStats, ProcessInfo, SystemCPU, SystemMEM, CpuCore): mô tả trạng thái hiện tại của hệ thống bao gồm mức sử dụng CPU, MEM và thông tin chi tiết của từng tiến trình.
3. Lớp giao tiếp mạng (TcpServer): đảm nhiệm việc mở cổng lắng nghe và tiếp nhận dữ liệu từ ứng dụng Linux. TcpServer phát tín hiệu dataReceived(...) mỗi khi nhận được dữ liệu mới, và lớp SystemMonitor sẽ tiếp nhận để xử lý tiếp.
4. Lớp xử lý dữ liệu (DataProcessor): độc lập và có thể tái sử dụng, chịu trách nhiệm phân tách chuỗi dữ liệu nhận được (dạng JSON hoặc tương đương) thành các đối tượng SystemStats và ProcessInfo. Việc tách lớp xử lý dữ liệu giúp dễ kiểm thử.
5. Lớp phát hiện quá tải (OverloadDetector): tiếp nhận dữ liệu hệ thống và kiểm tra theo ngưỡng cấu hình. Khi phát hiện trạng thái quá tải kéo dài, lớp này phát tín hiệu overloadDetected() giúp hệ thống kích hoạt hành động tương ứng như cảnh báo người dùng, lưu bản ghi chi tiết hoặc gửi lệnh đóng tiến trình thông qua LinuxApp.
6. Lớp lưu trữ (DatabaseManager, HistoryManager): chịu trách nhiệm ghi dữ liệu định kỳ vào các file Excel (CSV), lưu lại lịch sử hệ thống để phục vụ phân tích sau này. HistoryManager sử dụng DatabaseManager như một thành phần tổng hợp (aggregation) để thực hiện lưu/đọc dữ liệu.
7. Lớp cấu hình (ConfigManager): quản lý cấu hình toàn bộ hệ thống qua file **.ini**. Cung cấp các hàm getValue() và setValue() để các lớp khác sử dụng cấu hình một cách thống nhất.
8. Lớp hiển thị (ViewModel): gồm SystemStastViewModel, ProcessListModel, HistoryViewModel, các lớp này đóng vai trò là cầu nối giữa C++ và QML. Dữ liệu trong các ViewModel được cung cấp dưới dạng Q\_PROPERTY để hỗ trợ binding với giao diện QML theo chuẩn MVVM. Việc tách ViewModel giúp giao diện có thể cập nhật theo thời gian thực mà không cần biết chi tiết về tầng xử lý.



*Hình 4.4.2: Thiết kế đối tượng trên Windows App*

**5. Kế hoạch thực hiện và phân công:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Đảm nhiệm chính:**  **Hồ Đức Vũ** | **Component** | **From** | **To** | **Duration** |
| Build Documentation | Phân tích requirement | 12/3/2025 | 13/3/2025 | 1 tuần |
| Vẽ Usecase | 14/3/2025 | 15/3/2025 |
| Vẽ Class Diagram | 16/3/2025 | 17/3/2025 |
| Xây dựng ứng dụng thu thập thông số máy tính | Module thu thập thông số CPU: Tổng hệ thống và của từng core | 18/3/2025 | 19/3/2025 | 2,5 tuần |
| Module thu thập thông số MEM | 20/3/2025 | 20/3/2025 |
| Module thu thập thông số Network | 21/3/2025 | 21/3/2025 |
| Module thu thập thống số Disk I/O | 21/3/2025 | 21/3/2025 |
| Module thu thập thông số của các tiến trình: %CPU, MEM, Disk I/O | 22/3/2025 | 24/3/2025 |
| Module lọc các tiến trình "phù hợp"  1. Tạo Datasheet Command của các tiến trình , sử dụng trong Windows App 2. Tạo bộ lọc dự phòng (trong trường hợp Command không tồn tại trong Datasheet và Windows App gửi về thông điệp "Không xác định loại tiến trình" | 24/3/2025 | 26/3/2025 |
| Tích hợp các module vào hệ thống lớn | 27/3/2025 | 28/3/2025 |
| Module truyền dữ liệu đến Windows App | 29/3/2025 | 5/4/2025 |
| Xây dựng các hành động xử lý quá tải | Module xử lý terminate các tiến trình khi bị quá tải | 6/4/2025 | 7/4/2025 | 1 tuần |
| Module cảnh báo khi chạm ngưỡng tải (phát âm thanh cảnh báo thông qua PC speaker) | 8/4/2025 | 9/4/2025 |
| Đóng gói Linux App | Thiết lập install.sh, hoàn thành sản phẩm release (bản 1) Linux app | 10/4/2025 | 13/4/2025 |
| Xây dựng các trường hợp phát hiện quá tải hệ thống đơn giản | Module thiết lập rules cho các trường hợp quá tải toàn bộ hệ thống | 14/4/2025 |  | 1,5 tuần |
| Module đánh giá tải toàn bộ hệ thống dựa trên các rules đã thiết lập |  |  |
| Module thiết lập rules cho các trường hợp ngưỡng tải của các tiến trình |  |  |
| Module đánh giá tải của các tiến trình dựa trên các rules đã thiết lập |  |  |
| Module phát hiện chạm ngưỡng quá tải hệ thống |  |  |
| Tích hợp các module vào hệ thống lớn |  | 25/4/2025 |
| Xây dựng các hành động xử lý quá tải | Module lọc tiến trình hệ thống và tiến trình thường | 26/4/2025 |  | 1,5 tuần |
| Module xử lý khi quá tải nghiêm trọng (xử lý riêng biệt các trường hợp quá tải) |  |  |
| Module gửi tín hiệu ghi LOG đến Windows App |  |  |
| Tích hợp các module vào hệ thống lớn |  |  |
| Module các trường hợp quá tải toàn hệ thống |  | 7/5/2025 |
| Phát triển thuật toán phát hiện quá tải hệ thống và xử lý kết thúc các tiến trình trên hệ thống | Module ngưỡng tải của các tiến trình | 8/5/2025 |  | 2 tuần |
| Phát triển thuật toán kết thúc các tiến trình để tối ưu tải cho hệ thống và hạn chế ảnh hưởng đến trải nghiệm người dùng |  |  |
| <Xây dựng và cải thiện thông qua đánh giá> |  | 22/5/2025 |
| Stress test | Thiết lập ngưỡng tải tối đa (mô phỏng trên laptop) | 23/5/2025 |  | 1 tuần |
| Ghi log các thông số tải tối đa của hệ thống |  |  |
| Testing chức năng phát hiện và xử lý quá tải trong quá trình stress test |  | 30/5/2025 |
| Kiểm tra và đánh giá hệ thống | Kiểm tra từng module | 31/5/2025 |  | 1 tuần |
| Tích hợp toàn bộ module vào hệ thống lớn |  |  |
| Kiểm tra tổng thể hệ thống |  | 6/6/2025 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Đảm nhiệm chính:**  **Nguyễn Minh Phương** | **Component** | **From** | **To** | **Duration** |
| Build Documentation | Phân tích requirement | 12/3/2025 | 13/3/2025 | 1 tuần |
| Vẽ Usecase | 14/3/2025 | 15/3/2025 |  |
| Vẽ Class Diagram | 16/3/2025 | 17/3/2025 |  |
|  |  |  |  |  |
| Xây dựng giao diện hiển thị | (Tạm thời dùng dữ liệu giả) | 18/3/2025 | 1/4/2025 | 2 tuần |
|  | Hiển thị tải chiếm dụng của từng tiến trình | 19/3/2025 | 20/3/2025 |  |
|  | Hiển thị line chart với CPU | 21/3/2025 | 22/3/2025 |  |
|  | Hiển thị line chart với RAM | 23/3/2025 | 24/3/2025 |  |
|  | Sắp xếp ứng dụng theo mức sử dụng tài nguyên | 29/3/2025 | 30/3/2025 |  |
|  | Hiển thị lịch sử tải sử dụng |  |  |  |
| Module xử lý dữ liệu nhận từ app linux |  |  |  | 2 tuần |
|  | Viết module kết nối với Linux app | 31/3/2025 | 2/4/2025 |  |
|  | Nhận dữ liệu CPU -> phân tích rồi hiển thị lên UI | 3/4/2025 | 4/4/2025 |  |
|  | Nhận dữ liệu RAM -> phân tích rồi hiển thị lên UI |  |  |  |
|  | Nhận dữ liệu các tiến trình -> phân tích và hiển thị lên UI |  |  |  |
|  | Lưu dữ liệu nhận được vào file excel |  |  |  |
|  | Lưu dữ liệu nhận được trong thời gian quá tải vào file excel 2 |  |  |  |
| Module thông báo |  | 10/4/2025 | 20/4/2025 | 1 tuần |
|  | Hiển thị Pop-up cảnh báo khi quá tải |  |  |  |
| Các chức năng khác |  |  |  |  |
|  | Buộc đóng ứng dụng từ windown app |  |  |  |
|  | Tạo record dữ liệu những lần mà hệ thống xảy ra quá tải -> đánh giá các ứng dụng chiếm nhiều tài nguyên và thời gian quá tải trong bao lâu |  |  |  |
|  | Tạo record dữ liệu theo giờ, ngày, tháng,... () |  |  |  |

**Chú thích phân công trong bảng kế hoạch:**

|  |
| --- |
| **Cá nhân** |
| **Chung** |

Quá trình viết báo cáo xuyên suốt cả thời gian thực hiện đồ án, kể từ tháng 6 sẽ là thời gian tổng kết và đánh giá tổng thể.

**6. Tiến trình hiện tại**

Hiện tại tiến độ dự án đang đi đúng với kế hoạch đã đề ra. Chúng tôi đã phân tích đề tài, thiết kế UML bao gồm: Use case diagram, sequence diagram, activity diagram và class diagram, mức hoàn thành phần thiết kế phần mềm là 90%, 10% còn lại thuộc về phần chỉnh sửa và cũng như những phần mở rộng mà chúng tôi dự định sẽ phát triển cho phần mềm trong tương lai.

Đối với Linux App, chúng tôi đã hoàn thiện được 75% chức năng bao gồm: thu thập dữ liệu tài nguyên của toàn hệ thống và của các tiến trình đang hoạt động, thực hiện kết thúc các tiến trình theo command và phát âm cảnh báo thông qua Speaker. Đối với triển khai chức năng socket để kết nối truyền/nhận dữ liệu với Window App thì đã hoàn thành thiết kế class các chức năng cơ bản, mức độ hoàn thành chức năng này là 30%.

Đối với Window App, về phần UI, chúng tôi đã xây dựng được giao diện hiển thị danh sách các tiến trình. Còn với việc xây dựng UI về biểu đồ, cũng như xây dựng các lớp chính phía Backend sẽ tiếp tục tiến hành vào các tuần tiếp theo.