HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1

_____ o0o _____

BÀI TẬP LỚN

Đề tài 10: Tìm hiểu tổng quan về hệ điều hành thời gian thực (RTOS)

Môn học: Hệ điều hành
Số thứ tự nhóm: 04

Vũ Huy Hoàng	**MSSV: B21DCCN398**
Nguyễn Đức Hiếu	**MSSV: B21DCAT089**
Nguyễn Văn Mạnh	**MSSV: B21DCCN517**
Lê Đoàn Ngọc Nam	**MSSV: B21DCCN546**
Nguyễn Thế Độ	**MSSV: B21DCAT062**
Nguyễn Văn Trí	**MSSV: B21DCAT186**
Trần Hữu Đạt	**MSSV: B21DCCN221**

Giảng viên hướng dẫn: Ths. Đinh Xuân Trường

HÀ NỘI, 11/2023

LỜI CẢM ƠN

Trước tiên với tình cảm sâu sắc và chân thành nhất, cho phép chúng em được bày tỏ lòng biết ơn đến tất cả các cá nhân và tổ chức đã tạo điều kiện hỗ trợ, giúp đỡ nhóm chúng em trong suốt quá trình học tập và nghiên cứu đề tài này. Trong suốt thời gian từ khi bắt đầu học tập tại trường đến nay, chúng em đã nhận được rất nhiều sự quan tâm, giúp đỡ của các thầy cô và bạn bè.

Với lòng biết ơn sâu sắc nhất, nhóm chúng em xin gửi đến quý Thầy Cô ở Học viện Công nghệ Bưu Chính Viễn thông đã truyền đạt vốn kiến thức quý báu cho chúng em trong suốt quá trình học tập. Nhờ có những lời hướng dẫn, dạy bảo của các thầy cô nên đề tài nghiên cứu của chúng em mới có thể hoàn thiện tốt đẹp.

Một lần nữa, chúng em xin chân thành cảm ơn thầy Đinh Xuân Trường đã giúp đỡ, quan tâm để chúng em có thể hoàn thành tốt bài báo cáo này trong thời gian qua.

Bài báo cáo thực hiện trong khoảng thời gian hơn 1 tháng. Bước đầu đi vào thực tế của chúng em còn hạn chế và còn nhiều bỡ ngỡ nên không tránh khỏi những thiếu sót, chúng em rất mong nhận được những ý kiến đóng góp quý báu của thầy để kiến thức của nhóm chúng em trong lĩnh vực này được hoàn thiện hơn đồng thời có điều kiện bổ sung, nâng cao ý thức của mình.

Chúng em xin chân thành cảm ơn!

TÓM TẮT NỘI DUNG ĐỀ TÀI

RTOS (Real-time Operating System - Hệ điều hành thời gian thực) đang ngày càng phổ biến trong các sản phẩm công nghệ nhờ khả năng phục vụ các ứng dụng thời gian thực, với khả năng xử lý dữ liệu đầu vào nhanh chóng do không có sự chậm trễ của bộ đệm. Vì thế, mục tiêu của đề tài "Tìm hiểu tổng quan về hệ điều hành thời gian thực (RTOS)" là đưa ra cái nhìn tổng quan về lịch sử và quá trình phát triển của RTOS, lý do các nhà phát triển lựa chọn hệ điều hành này. Sau đó cụ thể hơn về các thành phần chức năng của RTOS như: quản lý file, quản lý bộ nhớ, quản lý tiến trình, các dịch vụ cung cấp bởi RTOS và so sánh ưu nhược điểm giữa RTOS và GPOS, từ đó phân tích một mô hình thực tế sử dụng RTOS. Sau cùng, đề tài sẽ giúp nhóm sinh viên và người đọc nắm rõ những kiến thức cơ bản về RTOS.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

Thuật ngữ	Ý nghĩa
ADAS	Hệ thống hỗ trợ người lái nâng cao (Advanced Driver Assistance Systems)
API	Giao diện lập trình ứng dụng (Application Programming Interface)
CAGR	Tỷ lệ tăng trưởng kép hàng năm (Compound Annual Growth Rate)
CPU	Bộ xử lý trung tâm (Central Processing Unit)
DBMS	Hệ thống quản lý cơ sở dữ liệu (Database Management System)
DMA	Bộ cấp phát bộ nhớ động (Dynamic Memory Allocator)
EDF	Thuật toán lập lịch theo hạn chót sớm nhất (Earliest Deadline First)
GERTS	General Electric Real-Time Executive
GPOS	Hệ điều hành đa mục đích (General Purpose Operating System)
IoT	Mạng lưới vạn vật kết nối Internet (Internet of Things)
IT	Công nghệ thông tin (Information Technology)
OS	Hệ điều hành (Operating SystemSystem)
PID	Thuật toán chuyển hóa tỷ lệ liên đới (Proportional Integral Derivative)
RAM	Bộ nhớ truy cập ngẫu nhiên (Random Access Memory)
RTC	Đồng hồ thời gian thực (Real-Time Clock)
RTOS	Hệ điều hành thời gian thực (Real-time Operating System)
SD	Chuẩn bảo mật kỹ thuật số (Secure Digital)

Thuật ngữ	**Ý nghĩa**
SPI	Giao diện kết nối ngoại vi (Serial Peripheral Interface)
UAV	Thiết bị bay không người lái (Unmanned Aerial Vehicle)
UAVCAN	Mạng khu vực điều khiển thiết bị bay không người lái (Unmanned Aerial Vehicle Controller Area Network)
VRTX	Virtual Real-Time Executive

CHƯƠNG 1. GIỚI THIỆU CHUNG VỀ RTOS

1.1 Tổng quan về RTOS

RTOS (Real-time Operating System - Hệ điều hành thời gian thực) là một hệ điều hành (OS) nhằm phục vụ các ứng dụng thời gian thực, với việc xử lý dữ liệu đầu vào mà không có sự chậm trễ của bộ đệm (buffer). Các yêu cầu về thời gian xử lý (bao gồm cả sự chậm trễ của hệ điều hành) được tính bằng phần mười của giây hoặc bằng thời gian ngắn hơn nữa. Quá trình xử lý trong RTOS phải diễn ra trong các ràng buộc, giới hạn thời gian được chỉ định, nếu không sẽ dẫn đến lỗi hệ thống.

RTOS hoạt động dựa trên hai cơ chế là hướng sự kiện (event-driven) hoặc chia sẻ thời gian (time-sharing):

- Cơ chế hướng sự kiện sẽ giải quyết và điều phối các tác vụ (task) thông qua mức độ ưu tiên của chúng.
- Cơ chế chia sẻ thời gian sẽ chuyển đổi các tác vụ dựa trên phản ứng ngắt của xung nhịp.

Phần lớn các hệ điều hành thời gian thực đều sử dụng giải thuật Pre-emptive Scheduling (tạm dịch: Lập lịch trước).

1.2 Lịch sử ra đời và phát triển

Hệ điều hành thời gian thực được phát triển lần đầu tiên vào những năm 1960 để đáp ứng nhu cầu của các hệ thống nhúng, nơi thời gian đáp ứng là rất quan trọng. Những hệ thống này thường được sử dụng trong các ứng dụng như điều khiển công nghiệp, y tế và an ninh.

Có thể chia lịch sử phát triển của RTOS thành 3 giai đoạn chính:

Giai đoạn 1 (1960-1970): Đây là giai đoạn đầu tiên của RTOS, với các hệ điều hành được viết bằng ngôn ngữ hợp ngữ và tối ưu hóa cho các phần cứng cụ thể. Một trong những RTOS đầu tiên là General Electric Real-Time Executive (GERTS), được phát triển vào đầu những năm 1960 cho máy tính GE 225. GERTS được thiết kế cho các ứng dụng điều khiển thời gian thực, chẳng hạn như tự động hóa công nghiệp và được sử dụng trong nhiều ngành công nghiệp khác nhau, bao gồm cả hàng không vũ trụ và quốc phòng.

Giai đoạn 2 (1970-1990): Đây là giai đoạn phát triển của RTOS, với các hệ điều hành được viết bằng ngôn ngữ lập trình cao cấp và trở nên linh hoạt hơn. Một số RTOS khác đã được phát triển, bao gồm cả hệ điều hành RT-11, được sử dụng trên máy tính mini PDP-11 của Digital Equipment Corporation. RT-11 được thiết

kế để xử lý các ứng dụng thời gian thực, chẳng hạn như điều khiển quá trình và thu thập dữ liệu, đồng thời được sử dụng rộng rãi trong nhiều ngành công nghiệp khác nhau.Vào những năm 1980, sự xuất hiện của bộ vi xử lý đã dẫn đến sự phát triển của RTOS cho các hệ thống nhúng. Một trong những RTOS đầu tiên dành cho hệ thống nhúng là VRTX (Virtual Real-Time Executive), được phát triển bởi Hunter & Ready Inc vào năm 1982. VRTX được thiết kế để sử dụng trên các bộ vi xử lý 8 bit và 16 bit và được sử dụng trong nhiều ứng dụng khác nhau, bao gồm hệ thống ô tô và điện tử tiêu dùng.

Giai đoạn 3 (1990-nay): Đây là giai đoạn hiện đại của RTOS, với các hệ điều hành thời gian thực trở nên phổ biến hơn và được sử dụng trong một loạt các ứng dụng. Vào những năm 1990, sự phát triển của các bộ vi xử lý mạnh hơn và sự phát triển của Internet đã dẫn đến sự phát triển của các RTOS phức tạp hơn, chẳng hạn như VxWorks và QNX. Các hệ thống vận hành toàn vẹn này được thiết kế để xử lý các ứng dụng thời gian thực phức tạp hơn, chẳng hạn như viễn thông, định tuyến mạng và đa phương tiện. Ngày nay, RTOS được sử dụng trong nhiều ngành công nghiệp khác nhau, bao gồm ô tô, hàng không vũ trụ, quốc phòng, viễn thông và điện tử tiêu dùng. Chúng là một phần thiết yếu của nhiều ứng dụng điều khiển thời gian thực, cung cấp môi trường đáng tin cậy và có thể dự đoán được để thực hiện các nhiệm vụ quan trọng.

1.3 Thị trường RTOS

1.3.1 Quy mô thị trường RTOS

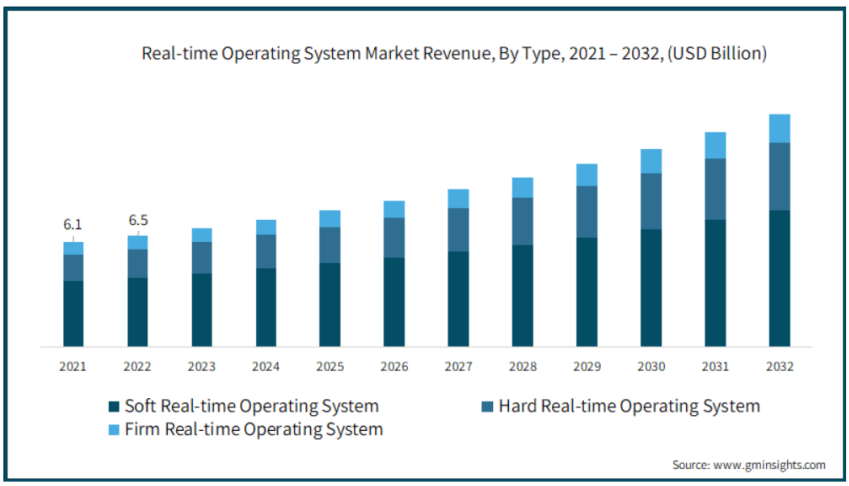
Hình 1.1: Nhìn chung về thị trường RTOS

Quy mô thị trường RTOS được định giá 6,5 tỷ USD vào năm 2022 và dự kiến sẽ tăng trưởng với tốc độ CAGR (Compound Annual Growth Rate - tỷ lệ tăng trưởng kép hàng năm) trên 7,5% từ năm 2023 đến năm 2032. Ngành này đang có đà tăng trưởng do nhu cầu về thiết bị điện tử ô tô ngày càng tăng. Ngành công nghiệp ô

tô đang ngày càng phụ thuộc vào các hệ thống điện tử về an toàn, thông tin giải trí (tạm dịch từ: infotainment, nghĩa là các chương trình truyền hình, tin tức,...được trình bày theo cách giúp người xem giải trí) và lái xe tự động. RTOS đóng một vai trò quan trọng trong việc quản lý các tác vụ theo thời gian thực trên xe, đảm bảo kiểm soát nhanh chóng các chức năng quan trọng về an toàn và tạo điều kiện tích hợp Hệ thống hỗ trợ người lái nâng cao (Advanced Driver Assistance Systems - ADAS).

1.3.2 Phân tích thị trường RTOS

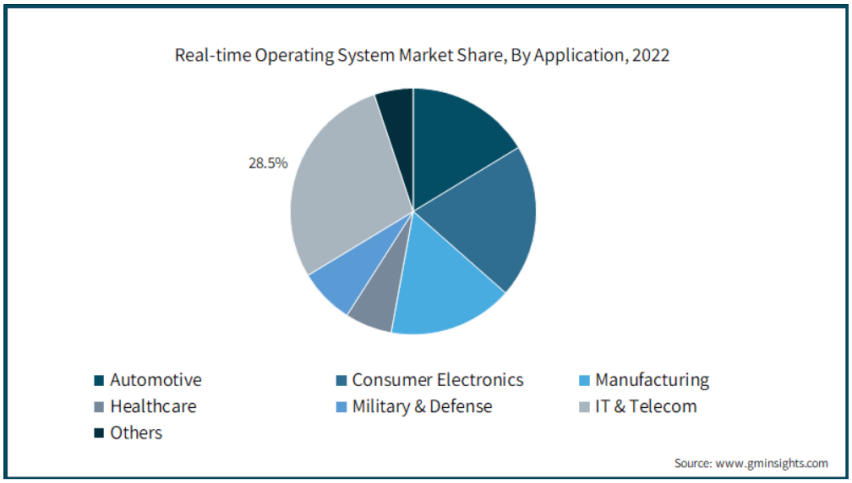
Dựa trên phân loại RTOS, thị trường RTOS được chia thành: soft RTOS, hard RTOS và firm RTOS.



Hình 1.2: Doanh thu và thị trường RTOS, theo loại, 2021-2032 (dự báo năm 2023 - 2032)

Phân khúc sorf RTOS được dự kiến sẽ tăng trưởng với tốc độ CAGR khoảng 6,5% trong giai đoạn dự báo. Soft RTOS đang có được sức hút nhờ tính linh hoạt và khả năng thích ứng, đặc biệt trong các ứng dụng mà các ràng buộc "chính xác tuyệt đối" về thời gian thực không quá quan trọng. Ví dụ như hệ thống thông tin giải trí ô tô và thiết bị nhà thông minh thể hiện xu hướng này. Soft RTOS cho phép sử dụng tài nguyên hiệu quả hơn, cho phép đa nhiệm liền mạch và xử lý khối lượng công việc tốt hơn. Hơn nữa, khi các thiết bị IoT tiếp tục phát triển, các giải pháp soft RTOS sẽ phù hợp hơn để quản lý các mức độ yêu cầu thời gian thực khác nhau, khiến chúng trở thành lựa chọn ưu tiên cho các nhà phát triển muốn cân bằng khả năng đáp ứng với hiệu quả và tính linh hoạt của hệ thống.

Dựa trên ứng dụng, thị trường hệ điều hành thời gian thực được phân loại thành Automotive (ô tô), Consumer Electronics (điện tử tiêu dùng), Manufacturing (sản xuất), Healthcare (chăm sóc sức khỏe), Military & Defense (quân sự & quốc phòng), IT & Telecom (công nghệ thông tin & viễn thông), v.v...



Hình 1.3: Thị phần RTOS theo ứng dụng, 2022

Mảng công nghệ thông tin & viễn thông được định giá hơn 1,5 tỷ USD vào năm 2022. RTOS đang có nhu cầu cao về các ứng dụng công nghệ thông tin và viễn thông do nhu cầu quan trọng của lĩnh vực này về khả năng đáp ứng theo thời gian thực. Trong viễn thông, RTOS rất quan trọng để quản lý thiết bị mạng, đảm bảo độ trễ thấp và độ tin cậy cao trong truyền tải dữ liệu. Ví dụ, việc sử dụng RTOS được thể hiện rõ trong hoạt động của các trạm cơ sở di động. Trong công nghệ thông tin, RTOS đóng vai trò quản lý các tác vụ như xử lý gói dữ liệu, bảo mật và quản lý mạng, hỗ trợ hiệu suất mạnh mẽ của bộ định tuyến, bộ chuyển mạch và các thiết bị mạng khác. Nhu cầu ngày càng tăng về các dịch vụ dữ liệu nhanh và không bị gián đoạn càng làm tăng thêm tầm quan trọng của RTOS trong các lĩnh vực này.

1.4 Lý do nên sử dụng RTOS

Hệ điều hành thời gian thực ngày càng trở thành một phần không thể thiếu trong nhiều hệ thống nhúng và sản phẩm công nghệ. Bằng cách tận dụng sức mạnh của RTOS, các nhà phát triển có thể tin tưởng rằng kiến trúc ứng dụng của họ sẽ đáng tin cậy và có thể đáp ứng các yêu cầu khắt khe nhất.

Dưới đây là một số lợi ích của việc sử dụng RTOS:

- Lập kế hoạch dựa trên mức độ ưu tiên: Các nhiệm vụ có thể được sắp xếp theo thứ tự quan trọng, đảm bảo chúng được xử lý hiệu quả và nhanh chóng. Điều này giúp cải thiện khả năng phản hồi của hệ thống, vì những nhiệm vụ quan trọng hơn sẽ được ưu tiên hơn những nhiệm vụ ít cần thiết hơn. Ngoài ra, việc lập lịch trình này đảm bảo không có nhiệm vụ nào bị dở dang do thiếu tài nguyên hoặc bộ nhớ. Do đó, các quy trình có thể chạy trơn tru mà không có bất kỳ sự chậm trễ hoặc gián đoạn nào. Hơn nữa, với việc lập kế hoạch dựa trên mức độ ưu tiên, các công ty có thể kiểm soát tốt hơn những nhiệm vụ nào được thực hiện trước, giúp bạn quản lý các hoạt động phức tạp dễ dàng hơn.

- Tách biệt thông tin thời gian (Abstracting Timing Information): RTOS có thể tách độc lập các tác vụ với thời gian chạy các tác vụ đó. Nghĩa là khi tác vụ chưa được hoàn thành, nhưng thời gian được thiết lập dành cho tác vụ đó đã hết, RTOS sẽ ngắt tác vụ nhờ cơ chế ngắt và chuyển qua xử lý các tác vụ khác. Điều này có ý nghĩa rất lớn, khi mỗi tác vụ sẽ được thiết lập một thời gian chạy cụ thể, độc lập với các tác vụ khác, giúp lập trình viên có thể dễ dàng tính toán phân tích và thiết kế hệ thống mà không phải lo lắng về vấn đề trễ hạn (deadline) hoặc tắc nghẽn tiềm ẩn trong hệ thống. Bên cạnh đó, lập trình viên có thể dễ dàng gỡ lỗi một cách nhanh chóng và hiệu quả nhờ các cơ chế ngắt, bộ xử lý lỗi. Mỗi tác vụ đều được thiết lập và thực thi riêng rẻ, không ảnh hưởng đến nhau. Điều này giúp các ứng dụng trở nên ổn định và mượt mà hơn do ít bị ảnh hưởng bởi biến về động về sức mạnh xử lý hoặc khối lượng công việc.
- Tính ổn định: RTOS có thể ưu tiên các quy trình nhất định đồng thời tránh độ trễ về thời gian hoặc tình trạng quá tải hệ thống do thiếu tài nguyên hoặc bộ nhớ. Điều này cho phép người dùng hoàn toàn tin tưởng vào hiệu suất ứng dụng của họ, đảm bảo rằng mọi tác vụ đều được hoàn thành đúng thời hạn mà không bị gián đoạn.
- Tính mô đun và khả năng bảo trì/nâng cấp: API dựa trên nhiệm vụ của RTOS khuyến khích phát triển mô-đun, nghĩa là các ứng dụng có thể được chia thành các nhiệm vụ riêng lẻ. Điều này làm cho mỗi nhiệm vụ dễ dàng hơn để phát triển và duy trì. Với RTOS, có thể dễ dàng tùy chỉnh nó để đáp ứng nhu cầu cụ thể của mình bằng cách thêm hoặc xóa các tính năng và thành phần nếu cần. Điều này giúp việc nâng cấp và bảo trì giải pháp trở nên dễ dàng hơn vì chỉ những phần cần thiết mới được cài đặt.
- Tận dụng và phát triển khả năng làm việc nhóm: Hệ thống dựa trên nhiệm vụ của RTOS cũng cho phép các nhà thiết kế/nhóm làm việc độc lập trên các phần dự án của họ. Điều này khuyến khích sự hợp tác và tăng năng suất. Nó cũng thúc đẩy tinh thần đồng đội bằng cách cho phép nhiều người làm việc cùng lúc trên các phần khác nhau của dự án. Với RTOS, nhiệm vụ có thể được phân chia giữa nhiều thành viên trong nhóm trong khi vẫn được quản lý theo thời gian thực. Điều này cho phép các dự án được hoàn thành nhanh hơn và hiệu quả hơn so với khi chúng được xử lý thủ công.
- Kiểm tra dễ dàng hơn: Phát triển dựa trên nhiệm vụ mô-đun dẫn đến thử nghiệm dựa trên nhiệm vụ mô-đun. Điều này giúp việc kiểm tra ứng dụng và xác định mọi lỗi tiềm ẩn trong mã trở nên dễ dàng hơn. Ưu điểm chính của

việc sử dụng RTOS là nó giúp việc kiểm tra dễ dàng hơn. Không giống như các hệ điều hành truyền thống, nó có thể được thử nghiệm trên các thiết bị và nền tảng khác nhau trong thời gian thực, từ đó có thể biết chính xác chúng sẽ hoạt động như thế nào khi được phát hành ra công chúng. Với RTOS, các nhà phát triển có quyền truy cập vào các công cụ mạnh mẽ giúp xác định và sửa lỗi nhanh hơn nhiều so với các hệ điều hành khác. Điều này làm cho chúng đáng tin cậy và hiệu quả hơn về lâu dài.

- Tái sử dụng mã: Với RTOS, các ứng dụng tương tự trên các nền tảng tương tự thường sẽ dẫn đến việc phát triển một thư viện các tác vụ tiêu chuẩn. Điều này có nghĩa là các nhà phát triển có thể sử dụng mã hiện có (chẳng hạn như trình điều khiển thiết bị) cho các dự án mới thay vì viết mọi thứ từ đầu. Điều này tiết kiệm thời gian và tiền bạc bằng cách cho phép các nhà phát triển tập trung vào các khía cạnh quan trọng hơn của dự án thay vì mất hàng giờ để viết lại mã.

Như vậy, RTOS mang lại rất nhiều lợi ích. Nhưng nói chung, các nhà phát triển nên cân nhắc sử dụng RTOS khi nhận thấy kiến trúc ứng dụng của họ cần đáp ứng những yêu cầu sau:

- Khi độ chính xác của thời gian xử lý là quan trọng: Nếu ứng dụng có các tác vụ phải hoàn thành trong một khung thời gian cụ thể, RTOS có thể giúp đảm bảo rằng các tác vụ đó được thực thi đúng thời hạn. Điều này đặc biệt quan trọng đối với các ứng dụng trong hệ thống điều khiển, robot và thiết bị y tế, nơi lỗi thời gian có thể gây ra hậu quả nghiêm trọng.
- Khi mức độ cạnh tranh tài nguyên cao: Trong các hệ thống nhúng có tài nguyên hạn chế (chẳng hạn như CPU, bộ nhớ và năng lượng) RTOS có thể quản lý hiệu quả các tài nguyên này để đảm bảo rằng các tác vụ không bị thiếu tài nguyên và có thể thực thi trơn tru.
- Khi khả năng phản hồi của hệ thống là quan trọng: Các ứng dụng yêu cầu phản hồi ngay lập tức đối với đầu vào của người dùng hoặc đọc cảm biến có thể được hưởng lợi từ khả năng ưu tiên và lập lịch tác vụ hiệu quả của RTOS. Điều này có thể cải thiện khả năng đáp ứng tổng thể của hệ thống và nâng cao trải nghiệm người dùng.
- Khi độ tin cậy hệ thống là tối quan trọng: RTOS cung cấp các cơ chế xử lý lỗi và khả năng chịu lỗi, khiến chúng phù hợp cho các ứng dụng mà lỗi có thể có tác động đáng kể.

CHƯƠNG 2. CÁC THÀNH PHẦN CHỨC NĂNG CỦA RTOS

2.1 Quản lý file

Quản lý file trong RTOS là việc quản lý các tập tin trên hệ thống file của RTOS. Hệ thống file của RTOS cung cấp các chức năng để tạo, mở, đọc, ghi, đóng, xóa và di chuyển các tập tin.

2.1.1 Phương pháp quản lý file trong RTOS

Có hai cách chính để quản lý file trong RTOS là sử dụng các hàm API và sử dụng thư viện file của bên thứ ba.

a, Sử dụng các hàm API của RTOS

Hàm API(Application Programming Interface – Giao diện lập trình ứng dụng) của RTOS là các hàm được cung cấp bởi hệ điều hành thời gian thực để cho phép lập trình viên tương tác với hệ thống. Các hàm API này cung cấp các chức năng cơ bản như tạo và quản lý tác vụ, quản lý bộ nhớ, giao tiếp giữa các tác vụ, và quản lý thời gian thực.

Các hàm API này thường được cung cấp dưới dạng các hàm C. Dưới đây là một số hàm API quản lý file phổ biến.

- *fopen()*: Tạo một file descriptor cho một tập tin. File descriptor trong RTOS là một số nguyên được sử dụng để tham chiếu đến một file hoặc một thiết bị. File descriptor được cấp phát bởi RTOS khi một file hoặc thiết bị được mở và được thu hồi khi file hoặc thiết bị được đóng.

```c
#include <stdio.h>

int main() {
  // Mở tệp tin để ghi
  FILE *file = fopen("data.txt", "w");
}

```

Hình 2.1: Ví dụ hàm *fopen()*

- *fread()*: Đọc dữ liệu từ một tập tin. Hàm này nhận bốn tham số:
  - file_descriptor: File descriptor của file hoặc thiết bị cần đọc.
  - buffer: Bộ đệm để lưu trữ dữ liệu được đọc.
  - size: Số lượng byte cần đọc.
  - offset: Vị trí trong file hoặc thiết bị cần bắt đầu đọc.

  Dưới đây là một ví dụ ngắn gọn về cách sử dụng hàm *fread()* để đọc một số byte từ đầu một file để mọi người hình dung dễ hơn:

```c
#include <stdio.h>

int main() {
  // Mở tệp tin để đọc
  FILE *file = fopen("data.txt", "r");
  if (file == NULL) {
    printf("Không thể mở tệp tin\n");
    return -1;
  }

  // Đọc dữ liệu từ tệp tin
  char buffer[100];
  size_t elements_read = fread(buffer, 1, sizeof(buffer), file);
  if (elements_read == 0) {
    printf("Lỗi khi đọc dữ liệu từ tệp\n");
    fclose(file);
    return -1;
  }

  // In dữ liệu đã đọc
  printf("Dữ liệu được đọc: %s\n", buffer);

  // Đóng tệp tin
  fclose(file);

  return 0;
}
```

Hình 2.2: Ví dụ hàm *fread()*

Trong ví dụ trên, hàm *fread()* sẽ đọc dữ liệu từ file *data.txt* và lưu trữ dữ liệu vào bộ đệm buffer. Số lượng byte được đọc sẽ được lưu trữ trong biến *bytes_read*. Nếu đọc thành công, hàm sẽ in dữ liệu được đọc ra màn hình.

- *fwrite()*: Ghi dữ liệu vào một tập tin. Hàm này nhận bốn tham số:
  - buffer: Bộ đệm chứa dữ liệu cần ghi.
  - size: Số lượng byte cần ghi.
  - offset: Vị trí trong file hoặc thiết bị cần bắt đầu ghi.
  - file_descriptor: File descriptor của file hoặc thiết bị cần ghi.

Dưới đây là một ví dụ về cách sử dụng hàm *fwrite()*:

```c
#include <stdio.h>
#include <string.h>
int main() {
  // Mở tệp tin để ghi
  FILE *file = fopen("data.txt", "w");
  if (file == NULL) {
    printf("Không thể mở tệp tin\n");
    return -1;
  }

  // Ghi dữ liệu vào tệp tin
  const char *data = "Hello, world!";
    size_t data_size = strlen(data);
  size_t elements_written = fwrite(data, 1, data_size, file);
  if (elements_written != data_size) {
    printf("Lỗi khi ghi dữ liệu vào tệp\n");
    fclose(file);
    return -1;
  }

  // Đóng tệp tin
  fclose(file);

  return 0;
}

```

Hình 2.3: Ví dụ hàm *fwrite()*

- *fclose()*: Đóng một file descriptor. Hàm này nhận một tham số là file descriptor cần đóng. Dưới đây là một ví dụ về cách sử dụng hàm *fclose()*:

```c
FILE *fp = fopen("myfile.txt", "r");
if (fp != NULL) {
  // Đọc dữ liệu từ tệp
  ...

  // Đóng tệp
  fclose(fp);
}

```

Hình 2.4: Ví dụ hàm *fclose()*

b, Sử dụng thư viện file của bên thứ ba

Ngoài các hàm API của RTOS, còn có nhiều thư viện file của bên thứ ba có thể sử dụng để quản lý file trong RTOS. Các thư viện file này thường cung cấp các chức năng quản lý file mạnh mẽ và linh hoạt hơn các hàm API của RTOS.

Một số thư viện file phổ biến bao gồm:

- FatFS: FatFS là một thư viện quản lý tệp dựa trên hệ thống tệp FAT (File Allocation Table). Nó cung cấp hỗ trợ cho hệ thống tệp FAT12, FAT16, và

FAT32 và là một thư viện phổ biến trong các ứng dụng nhúng. FatFS được sử dụng rộng rãi trên các thiết bị lưu trữ như thẻ SD (Secure Digital), USB (Universal Serical Bus), và các thiết bị lưu trữ flash.

- LittleFS: LittleFS là một thư viện quản lý tệp được tối ưu hóa cho các thiết bị lưu trữ flash, bao gồm cả flash NAND và flash SPI (Serial Peripheral Interface). Nó được phát triển đặc biệt cho các ứng dụng có tài nguyên và dung lượng nhỏ gọn.
- SDFat: Thư viện SDFat là một thư viện quản lý tệp dành riêng cho các thẻ SD và thiết bị lưu trữ flash khác. Thư viện này cung cấp các tính năng mạnh mẽ để làm việc với thẻ SD trong môi trường RTOS.

2.1.2 Quy trình quản lý file trong RTOS

a, Tạo một file descriptor cho tập tin cần quản lý

Để tạo một file descriptor cho tập tin cần quản lý trong RTOS, ta có thể sử dụng hàm *open()*. Dưới đây là ví dụ sử dụng hàm *open()* để tạo một file descriptor cho tập tin *data.txt* để đọc, ta có thể sử dụng mã sau:

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4
5 int main() {
6   // Tạo file descriptor
7   int file_descriptor = open("data.txt", O_RDONLY);
8
9 }
```

Hình 2.5: Sử dụng hàm *open()* để tạo một file descriptor cho tập tin *data.txt* để đọc

Ở ví dụ trên cờ *O_RDONLY* được sử dụng để mở file để đọc. Các cờ mở file thường được sử dụng bao gồm:

- *O_RDONLY*: Mở file để đọc.
- *O_WRONLY*: Mở file để ghi.
- *O_RDWR*: Mở file để đọc và ghi.

b, Thực hiện các thao tác quản lý file với file descriptor

Sau khi tạo một file descriptor cho tập tin cần quản lý, chúng ta có thể thực hiện các thao tác quản lý file với file descriptor này. Các thao tác quản lý file bao gồm:

- Đọc dữ liệu từ file: Sử dụng hàm *read()* để đọc dữ liệu từ file.
- Ghi dữ liệu vào file: Sử dụng hàm *write()* để ghi dữ liệu vào file.

- Xóa file: Sử dụng hàm *remove()* để xóa file.
- Tạo file mới: Sử dụng hàm *create()* để tạo file mới.
- Đổi tên file: Sử dụng hàm *rename()* để đổi tên file.
- Kiểm tra quyền truy cập file: Sử dụng hàm *access()* để kiểm tra quyền truy cập file.

c, Đóng file descriptor khi không còn cần sử dụng

Việc đóng file descriptor sẽ giúp giải phóng tài nguyên của hệ thống và tránh các lỗi tiềm ẩn.

Để đóng file descriptor, chúng ta sử dụng hàm *close()*. Hàm *close()* có một tham số là file descriptor cần đóng.

2.1.3 Định dạng file trong RTOS

Định dạng file trong RTOS là một vấn đề quan trọng, đặc biệt đối với các ứng dụng cần lưu trữ dữ liệu quan trọng hoặc cần truy cập dữ liệu nhanh chóng. Định dạng file tối ưu sẽ giúp cải thiện hiệu suất và độ tin cậy của hệ thống.

Dưới đây là một số định dạng file phổ biến được sử dụng trong RTOS:

- Tệp Nhị phân (Binary File): Tệp nhị phân chứa dữ liệu không được mã hóa dưới dạng văn bản. Đây có thể là tệp ảnh, tệp âm thanh, tệp thực thi của chương trình, hoặc dữ liệu nhị phân khác.
- Tệp Văn bản (Text File): Định dạng này chứa dữ liệu dưới dạng văn bản, thường được mã hóa bằng các mã như ASCII, UTF-8, hoặc UTF-16. Tệp văn bản thường chứa dòng văn bản, số, và các ký tự đặc biệt. Các tệp văn bản thường được sử dụng để lưu trữ cấu hình, tài liệu và dữ liệu dạng văn bản khác.
- Tệp CSV (Comma-Separated Values): Định dạng này được sử dụng để lưu trữ dữ liệu dạng bảng trong tệp văn bản. Các giá trị được phân cách bằng dấu phẩy hoặc các ký tự tương tự.
- Tệp JSON (JavaScript Object Notation): Định dạng này được sử dụng để lưu trữ và truyền tải dữ liệu có cấu trúc dưới dạng văn bản. JSON được sử dụng rộng rãi trong trao đổi dữ liệu giữa ứng dụng.
- Tệp XML (eXtensible Markup Language): Tệp XML chứa dữ liệu có cấu trúc dưới dạng văn bản, với cú pháp đánh dấu dựa trên thẻ. Định dạng XML thường được sử dụng để trao đổi dữ liệu cấu trúc.
- Tệp Log (Log File): Tệp log chứa dữ liệu được ghi lại từ các hoạt động hệ

thống hoặc ứng dụng. Chúng thường được sử dụng để theo dõi và gỡ lỗi.

- Tệp Cơ sở dữ liệu (Database File): Tệp cơ sở dữ liệu chứa dữ liệu được tổ chức và quản lý bằng Hệ thống quản lý cơ sở dữ liệu (DBMS - Database Management System). Định dạng cụ thể của tệp cơ sở dữ liệu phụ thuộc vào DBMS được sử dụng.

2.1.4 Những vấn đề khác

Ngoài những vấn đề đã đề cập ở trên, còn một số vấn đề khác liên quan đến quản lý file trong RTOS cần lưu ý, bao gồm:

a, Hiệu suất

Quản lý file cần được tối ưu hóa để đảm bảo hiệu suất của hệ thống. Một số kỹ thuật tối ưu hóa hiệu suất quản lý file bao gồm:

- Sử dụng bộ nhớ cache file.
- Tối ưu hóa các truy vấn file.
- Sử dụng các thuật toán truy cập file hiệu quả.

b, Tính bảo mật

Tính bảo mật trong RTOS là một vấn đề quan trọng RTOS cần phải được thiết kế và triển khai một cách an toàn để bảo vệ dữ liệu và hệ thống khỏi các cuộc tấn công:

- Sử dụng mã hóa file.
- Quản lý quyền truy cập file.
- Theo dõi hoạt động file.

c, Tính tương thích

Tính tương thích trong quản lí file RTOS là khả năng của các hàm quản lí file trên các RTOS khác nhau có thể sử dụng chung với nhau. Điều này giúp cho lập trình viên có thể viết mã quản lí file một lần và có thể sử dụng trên nhiều RTOS khác nhau mà không cần phải sửa đổi mã.

Dưới đây là một số lợi ích của tính tương thích trong quản lí file RTOS:

- Tiết kiệm thời gian và công sức: Lập trình viên chỉ cần viết mã quản lí file một lần và có thể sử dụng trên nhiều RTOS khác nhau.
- Tăng khả năng tái sử dụng mã: Mã quản lí file có thể được sử dụng trong nhiều ứng dụng khác nhau.
- Giảm thiểu rủi ro lỗi: Mã quản lí file được kiểm tra và thử nghiệm trên nhiều

RTOS khác nhau.

d, Khả năng mở rộng

Khả năng mở rộng trong quản lí file RTOS là khả năng của hệ thống quản lí file có thể được mở rộng để hỗ trợ các tính năng mới hoặc các thiết bị lưu trữ mới. Điều này giúp cho hệ thống quản lí file có thể đáp ứng được các yêu cầu của các ứng dụng mới hoặc các thiết bị lưu trữ mới.

Một số lợi ích của khả năng mở rộng trong quản lí file RTOS:

- Cho phép hệ thống quản lý file đáp ứng được các yêu cầu của các ứng dụng mới hoặc các thiết bị lưu trữ mới.
- Giúp tiết kiệm thời gian và công sức cho lập trình viên.

2.2 Quản lý bộ nhớ

Quản lý bộ nhớ trong RTOS là một khía cạnh quan trọng để đảm bảo rằng hệ thống có thể đáp ứng được yêu cầu thời gian thực của nó trong khi sử dụng tài nguyên bộ nhớ hiệu quả. RTOS được thiết để tối ưu hóa về mặt thời gian và xử lý các tác vụ 1 cách mượt mà, nhanh chóng. Do đó quản lý bộ nhớ hiệu quả là 1 yếu tố rất quan trọng để tránh các vấn đề như phân mảnh bộ nhớ, xung đột tài nguyên bộ nhớ giữa các thành phần phần cứng hoặc tác vụ, chậm trễ trong quá trình xử lý hoặc tệ nhất là có thể dẫn đến treo máy trên toàn bộ hệ thống.

Về cơ bản, quản lý bộ nhớ trong RTOS có thể chia thành 2 kỹ thuật chính là quản lý bộ nhớ cố định và quản lý bộ nhớ động. Bên cạnh đó, RTOS có 2 cách để quản lý bộ nhớ là Stack Memory và Heap Memory.

Stack Memory (Bộ nhớ ngăn xếp) là phần bộ nhớ được sử dụng trong quá trình vận hành Khối điều khiển tác vụ trung tâm (Task Control Blocks), bao gồm các thông tin gọi hàm, biến và dữ liệu điều khiển hệ thống.

Heap Memory (Bộ nhớ Heap) là bộ nhớ được sử dụng để cấp phát bộ nhớ động như lưu trữ các danh sách liên kết, cấu trúc dữ liệu, bitmap... trong các thuật toán quản lý bộ nhớ động (sẽ nói rõ ở phần sau).

2.2.1 Quản lý bộ nhớ cố định

Quản lý bộ nhớ cố định là kỹ thuật phân chia bộ nhớ tại thời điểm biên dịch. Mỗi tiến trình sẽ được xác định cụ thể 1 kích thước bộ nhớ cố định. Phần bộ nhớ này là không thể thay đổi trong suốt quá trình chạy của tiến trình đó, cho đến khi tiến trình kết thúc, bộ nhớ sẽ được thu hồi và cấp phát lại cho tiến trình khác theo 1 kích thước mới. Quản lý bộ nhớ theo cách này có thể giảm thiểu hiện tượng phân mảnh bộ nhớ và tăng hiệu năng xử lý của hệ điều hành do không phải thực hiện tái

phân bổ bộ nhớ trong quá trình hoạt động.

Tuy nhiên do lượng bộ nhớ là cố định, khi một tác vụ được kết thúc sớm hơn so với dự tính hoặc bị ngắt bởi một điều kiện đặc biệt, phần bộ nhớ được cấp phát cho tác vụ đó sẽ không được tái sử dụng cho đến khi hệ điều hành thực hiện lần phiên dịch tiếp theo. Điều này gây tốn và không tối ưu được lượng bộ nhớ dư thừa. Quản lý bộ nhớ cố định đòi hỏi lập trình viên phải tính toán và tối ưu mã nguồn cũng như các tiến trình xử lý các tính huống có thể xảy ra.

2.2.2 Quản lý bộ nhớ động

Quản lý bộ nhớ động là kỹ thuật phân chia bộ nhớ tài thời điểm chạy thay vì biên dịch. Sử dụng các thuật toán đặc biệt, RTOS sẽ tính toán và cấp phát 1 lượng bộ nhớ vừa đủ cho tiến trình đang chạy. Tiến trình kết thúc, bộ nhớ sẽ được thu hồi, tính toán và cấp phát lại cho tiến trình khác. Nhờ có cơ chế linh hoạt trong việc phân phát và thu hồi, quản lý bộ nhớ động giúp tiết kiệm đáng kể dung lượng bộ nhớ, giúp hệ điều hành có thể xử lý với các biến số không được chuẩn bị trước thông qua các cơ chế ngắt và xử lý lỗi. Quản lý bộ nhớ động có thể được chia thành 2 loại là quản lý thủ công và quản lý tự động.

Trong một số trường hợp, có thể do thiếu sót trong quá trình phát triển của lập trình viên dẫn đến thuật toán tái phân bổ hoạt động không hiệu quả, gây hiện tượng phân mảnh bộ nhớ. Nếu không kịp thời xử lý có thể dẫn diễn rò rỉ bộ nhớ. Mặt khác, mã nguồn của kỹ thuật quản lý bộ nhớ động không yêu cầu lập trình viên phải thiết lập các thông số bộ nhớ cố định của tác vụ, nhưng đòi hỏi lập trình viên phải thiết kế các bộ xử lý lỗi đủ tốt để kiểm soát được các tình huống bất thường. Vi xử lý phần cứng cũng phải đủ mạnh để có thể đồng thời thực hiện việc tái phân bổ bộ nhớ trong lúc các tác vụ khác đang được thực thi.

a, Quản lý thủ công

Quản lý thủ công là cách quản lý bộ nhớ chủ yếu dựa vào sự kiểm soát của lập trình viên. Các thao tác như cấp phát bộ nhớ, thu hồi và tái cấp phát, thực hiện xử lý ngắt, kiểm soát lỗi và các tình huống bất thường sẽ được thực hiện bởi lập trình viên. Cách này có thể giúp lập trình viên can thiệp sâu nhất vào hệ điều hành, từ đó quản lý chính xác tài nguyên bộ nhớ.

Tuy nhiên, nhược điểm của cách này là có thể dẫn đến các lỗi "con người" như quên giải phóng bộ nhớ đã cấp phát hoặc truy cập vào các bộ nhớ ảo (các bộ nhớ đã giải phóng nhưng lập trình viên quên mất điều đó), dẫn đến tình trạng con trỏ treo. Đôi khi tệ hơn là gây phân mảnh trong do thiếu sót trong tư duy, thao tác của lập trình viên.

b, Quản lý tự động

Ngược lại với quản lý thủ công, quản lý tự động là cách quản lý bộ nhớ bằng cách sử dụng các bộ thu gom rác. Bộ thu gom rác là các chương trình con được xây dựng dựa trên thuật toán quản lý bộ nhớ (sẽ trình bày ở phần sau) để thu gom các "bộ nhớ rác", là các phần bộ nhớ còn lại sau khi tác vụ được cấp phát phần bộ nhớ đó đã hoàn tất. Do được xây dụng dựa trên thuật toán tối ưu, quản lý tự động giúp giảm các lỗi "con người", tối ưu hóa quy trình tái phân bổ bộ nhớ của hệ thống.

Mặt khác, vì bản chất là thuật toán quản lý bộ nhớ, nên các chương trình con có thể gây ra lỗi nếu lập trình viên ứng dụng thuật toán không đúng cách hoặc lỗi lập trình, dẫn đến treo diện rộng trên toàn bộ hệ thống. Bên cạnh đó, bản thân các thuật toán cũng không phải là hoàn hảo, vẫn còn cần có sự giúp đỡ của lập trình viên giám sát trong quá trình vận hành, đảm bảo hệ thống luôn ổn định và hoạt động đúng cách. Yêu cầu về yếu tố phần cứng đối với quản lý tự động cũng cao hơn so với thủ công do hệ điều hành phải dành 1 phần hiệu năng để duy trì các chương trình phân bổ bộ nhớ luôn chạy, đảm bảo tài nguyên bộ nhớ luôn được tối ưu.

c, Các thuật toán cấp phát bộ nhớ động

Đa số các thuật toán cấp phát bộ nhớ động phổ biến đều dựa trên nền tảng của Bộ cấp phát bộ nhớ động (DMA – Dynamic Memory Allocator). Mục đích của DMA là phân phát các phần bộ nhớ có sẵn chưa được sử dụng cho các ứng dụng/tiến trình chưa được cấp phát bộ nhớ. Phương pháp chính thường được sử dụng là lưu 1 danh sách liên kết giữa các phần bộ nhớ đã được cấp phát và phần bộ nhớ khả dụng còn lại. Khi đó, mỗi khi 1 tiến trình cần cấp phát bộ nhớ, phần bộ nhớ trống được liên kết thông qua danh sách sẽ được sử dụng để cấp phát cho tiến trình đó. Dựa trên phương pháp cơ sở của DMA, các thuật toán cấp phát bộ nhớ động được tạo ra để tối ưu hiệu suất và tăng khả năng quản lý/phân phát bộ nhớ trong RTOS.

- **Sequential Fit (tạm dịch: Phù hợp tuần tự)**

  Thuật toán này xây dựng 1 danh sách liên kết đơn các vùng bộ nhớ trống. Các vùng bộ nhớ này sẽ được cấp phát cho tiến trình theo 4 cách khác nhau:

  – Phù hợp đầu tiên (First Fit): Danh sách liên kết sẽ được duyệt tuần tự từ đầu và trả về ô nhớ đầu tiên đủ lớn với yêu cầu của tiến trình.

  – Phù hợp tiếp theo (Next Fit): Danh sách liên kết sẽ được duyệt từ vị trí lần tìm kiếm cuối cùng dừng lại, sau đó trả về ô nhớ tiếp theo phù hợp với yêu cầu.

  – Phù hợp nhất (Best Fit): Danh sách liên kết sẽ được duyệt toàn bộ, sau đó trả về ô nhớ nhỏ nhất đủ lớn để phù hợp với yêu cầu.

  - Phù hợp tệ nhất (Worst Fit): Danh sách liên kết cũng sẽ được duyệt toàn bộ, nhưng sẽ trả về ô nhớ lớn nhất.

Nhờ ưu điểm đơn giản, Sequential Fit rất dễ áp dụng. Tuy nhiên, thời gian tìm kiếm ô nhớ phù hợp tỉ lệ thuận với kích thước bộ nhớ. Bộ nhớ càng lớn, thời gian tìm kiếm càng lâu. Trường hợp xấu hơn còn có thể tốn thêm bộ nhớ để duy trì danh sách liên kết, gây ra kết quả không mong muốn.

- **Buddy Allocator (tạm dịch: Phân phối kề cận)**

Buddy Allocator sử dụng một mảng các danh sách liên kết đơn, mỗi mảng đại diện cho 1 kích thước ô nhớ tương ứng. Khi nhận được yêu cầu cấp phát bộ nhớ, thuật toán sẽ tìm kiếm ô nhớ còn trống trong danh sách phù hợp với kích thước ô nhớ của yêu cầu. Nếu danh sách trống (tức là toàn bộ ô nhớ thuộc về danh sách đó đều đã được sử dụng) thì thuật toán sẽ tìm kiếm ô nhớ trống trong danh sách có kích thước lớn hơn, sau đó chọn một ô nhớ và chia đôi ô nhớ đó thành 2 phần bằng nhau. Theo hướng ngược lại, nếu không có ô nhớ còn trống nào đủ lớn cho yêu cầu của tiến trình, ô nhớ có thể hợp nhất với ô nhớ liền kề với nó nếu ô nhớ liền kề đó chưa bị chia thành 2 ô nhớ nhỏ hơn.

Ưu điểm của Phân phối kề cận là ô nhớ liền kề của 1 ô nhớ đang được cấp phát có thể được xác định một cách nhanh chóng. Tuy nhiên do số lượng kích thước ô nhớ là hữu hạn, phân mảnh trong có thể xuất hiện, gây lãng phí bộ nhớ không mong muốn.

- **Indexed Fit (tạm dịch: Chỉ mục phù hợp)**

Đối với thuật toán này, lập trình viên sử dụng các cấu trúc dữ liệu như Tree (cây) hoặc Hash Table (bảng băm) để lập chỉ mục cho các ô nhớ còn trống dựa trên chiến lược phân chương bộ nhớ tương ứng.

Trên lý thuyết, thuật toán này khá đơn giản và tối ưu. Tuy nhiên đòi hỏi lập trình viên phải có kiến thức và kĩ thuật đủ tốt để đưa ra được các chiến lược phân chương bộ nhớ tối ưu và tận dụng được thế mạnh của các cấu trúc dữ liệu.

- **Bitmapped Fit (tạm dịch: Ánh xạ địa chỉ bit phù hợp)**

Khi sử dụng Bitmapped Fit, RTOS sẽ thiết lập 1 bitmap (bản đồ bit) để thể hiện (ánh xạ) trạng thái sử dụng của bộ nhớ heap. Mỗi bit trên bitmap sẽ thể hiện một phần của bộ nhớ Heap. Nếu phần đó đang được sử dụng, bit tương ứng sẽ có giá trị là 1, ngược lại là 0.

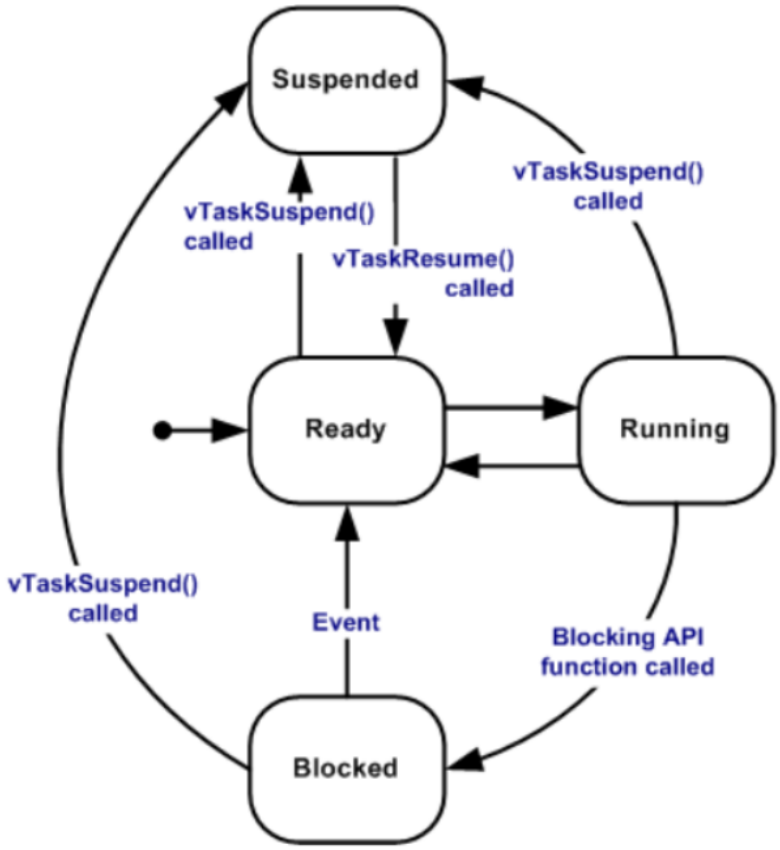
Do việc ánh xạ địa chỉ chỉ diễn ra trong một khoảng thời gian rất nhỏ, nên

thời gian tìm kiếm ô nhớ còn trống tỉ lệ thuận và phụ thuộc vào kích thước của bitmap được thiết lập.

2.3 Quản lý tiến trình

2.3.1 Các trạng thái của tiến trình

Trong RTOS có 4 trạng thái tiến trình bao gồm Chạy, Sẵn sàng, Bị chặn và Đình chỉ.



Hình 2.6: Nguyên tắc hoạt động của các trạng thái tiến trình

Chạy (Running): Khi một tác vụ thực sự được thực thi, nó được cho là ở trạng thái đang chạy. Nếu bộ xử lý mà RTOS đang chạy chỉ có một lõi (single core) thì chỉ có thể có một tác vụ ở trạng thái đang chạy tại bất kỳ thời điểm nào. Điều này là do một tác vụ ở trạng thái thực thi có quyền kiểm soát bộ xử lý cơ bản.

Sẵn sàng (Ready): Một tác vụ ở trạng thái sẵn sàng khi nó sẵn sàng để thực thi (không ở trạng thái bị chặn hoặc bị đình chỉ) nhưng hiện không thực thi vì một tác vụ khác có mức độ ưu tiên bằng hoặc cao hơn đã ở trạng thái đang chạy.

Bị chặn (Blocked): Một tác vụ ở trạng thái bị chặn sẽ không đủ điều kiện để lập lịch (scheduling). Một tác vụ sẽ ở trạng thái bị chặn bất cứ khi nào nó đang chờ một sự kiện xảy ra. Sự kiện này có thể đang hoàn thành một khoảng thời gian trì

hoãn hoặc sự sẵn có của một nguồn tài nguyên. Sau khi nguyên nhân gây ra khối được loại bỏ, tác vụ sẽ được đặt trở lại trạng thái sẵn sàng.

Đình Chỉ (Suspended): Tác vụ trong trạng thái này đã bị đình chỉ và không thể thực hiện tiếp. Điều này có thể xảy ra do yêu cầu của người dùng hoặc khi có sự cố xảy ra. Trạng thái này có thể dễ dàng chuyển đổi giữa các trạng thái khác.

2.3.2 Các thuật toán quản lý tiến trình trong RTOS

a, Lập lịch ưu tiên cố định (Fixed Priority)

Mỗi tiến trình hoặc tác vụ sẽ được gán một ưu tiên nhất định, tác vụ nào có ưu tiên cao hơn sẽ được thực hiện trước.

- Ưu điểm: Dễ triển khai, dễ hiểu, đảm bảo ưu tiên thực hiện các tiến trình quan trọng hơn.
- Nhược điểm: Không linh hoạt khi các ưu tiên cố định cần được thay đổi.

b, Vòng tròn (Round-Robin)

Trong thuật toán này, mỗi tiến trình được cấp một khoảng thời gian thực hiện cố định (quantum). Khi quantum kết thúc, CPU chuyển sang tiến trình khác trong hàng đợi. Điều này đảm bảo công bằng trong việc phân chia thời gian CPU.

- Ưu điểm: Công bằng trong phân chia thời gian CPU, ngăn chặn việc tiến trình chiếm giữ CPU vô hạn.
- Nhược điểm: Thời gian chuyển đổi giữa các tiến trình có thể gây ra độ trễ.

c, Theo hạn chót sớm nhất (Earliest Deadline First – EDF)

Thuật toán EDF ưu tiên thực hiện tiến trình có hạn chót(deadline) sớm nhất. Tiến trình có hạn chót sớm nhất sẽ được chọn để chạy tiếp theo.

- Ưu điểm: Phù hợp cho các ứng dụng thời gian thực với các hạn chót cứng.
- Nhược điểm: Yêu cầu tính toán và quản lý hạn chót, điều này có thể gây độ trễ hoặc tăng khả năng lỗi.

d, Theo tần số nhỏ nhất(Rate Monotonic)

Trong thuật toán này, các tiến trình có tần số thực hiện ngắn hơn sẽ được ưu tiên cao hơn. Điều này đảm bảo rằng các tiến trình có chu kỳ ngắn sẽ được thực hiện trước.

- Ưu điểm: Dễ triển khai và có tính ổn định cao, hiệu suất tốt cho các tác vụ có chu kỳ ngắn.
- Nhược điểm: Không linh hoạt cho các ứng dụng có các tác vụ có độ phức tạp khác nhau.

e, Theo hạn chót nhỏ nhất(Deadline Monotonic)

Trong thuật toán Deadline Monotonic, tiến trình với hạn chót gần nhất sẽ được ưu tiên cao hơn. Điều này phù hợp cho các ứng dụng thời gian thực với hạn chót động.

- Ưu điểm: Dễ triển khai và có tính ổn định cao, hiệu suất tốt cho các ứng dụng với các hạn chót động.
- Nhược điểm: Không linh hoạt cho các ứng dụng có tác vụ có độ phức tạp khác nhau.

2.4 Các loại dịch vụ cung cấp bởi RTOS

RTOS cung cấp nhiều dịch vụ và tính năng quan trọng để quản lý và điều khiển các ứng dụng thời gian thực. Dưới đây là một phân tích chi tiết về các dịch vụ chính được cung cấp bởi RTOS:

- Lập Lịch: Dịch vụ lập lịch là một trong những yếu tố quan trọng nhất của RTOS. Nó quyết định thứ tự thực hiện của các tác vụ hoặc tiến trình trong hệ thống. RTOS sử dụng các thuật toán lập lịch để quyết định xem tiến trình nào sẽ được thực hiện tiếp theo dựa trên ưu tiên và độ ưu tiên của chúng.
- Quản Lý Tài Nguyên: RTOS quản lý tài nguyên phần cứng của hệ thống như CPU, bộ nhớ, các cổng giao tiếp, và các thiết bị ngoại vi. Điều này bao gồm việc đảm bảo rằng một tài nguyên không được sử dụng bởi nhiều tiến trình cùng một lúc và nó được phân chia một cách hiệu quả.
- Bảo Vệ Dữ Liệu và Tiến Trình: RTOS cung cấp các cơ chế bảo vệ để đảm bảo tính nhất quán và bảo mật dữ liệu giữa các tiến trình hoặc tác vụ khác nhau. Điều này đặc biệt quan trọng trong các hệ thống đa nhiệm và đa người dùng.
- Quản Lý Thời Gian: RTOS cung cấp các chức năng để đo thời gian và đảm bảo tính chính xác thời gian cho các ứng dụng thời gian thực.Điều này bao gồm việc quản lý đồng hồ thời gian thực (RTC - Real-Time Clock) và tích hợp các tính năng đặc biệt để theo dõi thời gian.
- Giao Tiếp Trong Hệ Thống: RTOS cung cấp các cơ chế giao tiếp để cho phép các tiến trình hoặc tác vụ trao đổi thông tin và dữ liệu. Điều này có thể bao gồm các hàng đợi, xem ngắn, và cơ chế giao tiếp trực tiếp giữa các tiến trình.
- Xử Lý Ngoại Lệ và Sự Cố: RTOS có khả năng xử lý ngoại lệ và sự cố một cách an toàn. Nếu xảy ra lỗi hoặc sự cố, RTOS có thể cung cấp các cơ chế để đảm bảo rằng hệ thống không gặp sự cố nghiêm trọng và có thể phục hồi một cách an toàn.

- Lập Lịch Ưu Tiên: RTOS cho phép xác định độ ưu tiên của các tiến trình hoặc tác vụ. Điều này quan trọng để quyết định xem tiến trình nào được ưu tiên thực hiện khi cùng có nhiệm vụ cần được thực hiện.
- Dịch Vụ Đồng Bộ Hóa: RTOS cung cấp các cơ chế đồng bộ hóa để đảm bảo rằng các tiến trình hoặc tác vụ hoạt động một cách đồng bộ và không gây ra xung đột dữ liệu hoặc tình trạng cạnh tranh.
- Quản Lý Lỗi và Ghi Log: RTOS thường có cơ chế quản lý lỗi và ghi log để theo dõi và ghi lại các sự kiện quan trọng trong hệ thống. Điều này giúp trong việc phát hiện và khắc phục sự cố.
- Các Dịch Vụ Mạng (tuỳ theo RTOS): Một số phiên bản RTOS có tích hợp dịch vụ mạng, cho phép ứng dụng thời gian thực kết nối và giao tiếp trên mạng.

Các dịch vụ này tạo nên cơ sở để xây dựng và quản lý các ứng dụng thời gian thực trên một hệ thống sử dụng RTOS. Tùy thuộc vào phiên bản RTOS cụ thể, các dịch vụ và tính năng có thể khác nhau.

2.5 Phân loại và biến thể của RTOS

2.5.1 Phân loại RTOS

RTOS hoạt động dựa trên hai cơ chế là hướng sự kiện (event-driven) hoặc chia sẻ thời gian (time-sharing). Cơ chế hướng sự kiện sẽ giải quyết và điều phối các tác vụ (task) thông qua mức độ ưu tiên của chúng, còn cơ chế chia sẻ thời gian sẽ chuyển đổi các tác vụ dựa trên phản ứng ngắt của xung nhịp. Dựa vào cơ chế này, Hệ điều hành RTOS thường được chia thành ba loại chính là: hard RTOS, firm RTOS, và soft RTOS.

Trong hard RTOS, thời hạn được xử lý rất nghiêm ngặt, điều đó có nghĩa là nhiệm vụ nhất định phải bắt đầu thực hiện theo thời gian đã lên lịch đã chỉ định và phải hoàn thành trong khoảng thời gian được chỉ định. Ví dụ: Hệ thống chăm sóc quan trọng về mặt y tế, Hệ thống máy bay,...

Bên cạnh hard RTOS, firm RTOS cũng cần tuân thủ thời hạn. Tuy nhiên, việc trễ thời hạn có thể không ảnh hưởng lớn nhưng có thể gây ra những ảnh hưởng không mong muốn, chẳng hạn như chất lượng sản phẩm bị giảm đi rất nhiều. Ví dụ: Các loại ứng dụng đa phương tiện,...

Trong khi đó, soft RTOS chấp nhận một số độ trễ của Hệ điều hành. Trong loại RTOS này, có thời hạn được ấn định cho một công việc cụ thể nhưng có thể chấp nhận được sự chậm trễ trong một khoảng thời gian nhỏ. Vì vậy, thời hạn được loại RTOS này xử lý nhẹ nhàng.Ví dụ: Hệ thống giao dịch trực tuyến, Hệ thống báo giá chăn nuôi,...

2.5.2 Các biến thể của RTOS

RTOS có nhiều biến thể và phiên bản phát triển để đáp ứng các yêu cầu và nhu cầu khác nhau trong lĩnh vực hệ thống thời gian thực. Dưới đây là một số biến thể quan trọng của RTOS và mô tả tóm tắt về mỗi loại:

- freeRTOS: freeRTOS là một RTOS mã nguồn mở phổ biến và miễn phí. Nó được thiết kế cho các ứng dụng nhúng và là một trong những RTOS được sử dụng rộng rãi nhất trên thế giới.

  Đặc điểm: Nhẹ nhàng và tiết kiệm tài nguyên, dễ tích hợp vào các ứng dụng nhúng. Cung cấp các kiến thức lập lịch đơn và đa nhiệm đơn giản.Mã nguồn mở với giấy phép Apache, giúp phát triển ứng dụng miễn phí.

  Ứng dụng: Được sử dụng trong các hệ thống nhúng như thiết bị y tế, thiết bị IoT, và hệ thống nhúng đa dạng.

- VxWorks: VxWorks là một RTOS thương mại phổ biến được phát triển bởi Wind River Systems. Nó được sử dụng trong các ứng dụng quan trọng và yêu cầu độ tin cậy cao.

  Đặc điểm: Hỗ trợ cho nhiều kiến thức lập lịch đa mức và đa nhiệm. Quản lý tài nguyên mạnh mẽ và kiến thức giao tiếp cao. Được sử dụng rộng rãi trong các hệ thống nhúng phức tạp và yêu cầu độ tin cậy cao.

  Ứng dụng: Trong các ứng dụng y tế, hàng không, không gian, ô tô, và nhiều lĩnh vực quan trọng khác.

- QNX: QNX là một RTOS thương mại phát triển bởi QNX Software Systems (nay thuộc quyền sở hữu của BlackBerry). Nó được sử dụng rộng rãi trong các ứng dụng đòi hỏi độ ổn định và phản hồi nhanh.

  Đặc điểm: Hệ thống lõi thời gian thực mạnh mẽ với kiến thức tương tác cao và quản lý tài nguyên tốt. Hỗ trợ đa nhiệm và đa người dùng.

  Ứng dụng: Trong các ứng dụng trong lĩnh vực ô tô, y tế, sản xuất và điều khiển tự động.

- Các RTOS mã nguồn mở khác: Có nhiều RTOS mã nguồn mở khác như RTEMS, NuttX, ChibiOS/RT, và C/OS-II.

  Đặc điểm: Thường nhẹ nhàng và phù hợp cho các ứng dụng nhúng với tài nguyên hạn chế. Dựa trên cộng đồng phát triển và thường miễn phí.

- RTOS tích hợp mạng: Một số biến thể của RTOS có tích hợp sẵn các dịch vụ mạng, cho phép kết nối và giao tiếp qua mạng. Ví dụ như ThreadX và embOS

có phiên bản tích hợp mạng cho các ứng dụng IoT.

Đặc điểm: Hỗ trợ kết nối mạng và giao tiếp trong các ứng dụng IoT.

- RTOS cho ứng dụng nhỏ gọn: Các RTOS như TinyOS và Contiki được phát triển đặc biệt cho các ứng dụng IoT và hệ thống cảm biến. Chúng nhẹ nhàng và tiêu tốn ít tài nguyên.

Đặc điểm: Phù hợp cho các hệ thống nhúng nhỏ gọn và tiêu thụ ít năng lượng.

2.6 So sánh RTOS và GPOS

2.6.1 Điểm khác biệt của RTOS so với GPOS

- Các tác vụ của RTOS phải luôn bị bắt buộc phải xử lý trong thời gian quy định để đảm bảo không có lỗi nghiêm trọng nào xảy ra.
- RTOS luôn luôn lập lịch các tiến trình dựa trên mức độ ưu tiên của chúng. Điều này là không được khi ở trong GPOS.
- Luồng có mức ưu tiên cao hơn có thể yêu cầu luồng đang chạy bị ngừng, cho dù luồng đó có đang thực hiện lệnh gọi trong lõi hệ điều hành. Điều này là không được khi ở trong GPOS.
- RTOS được thiết kế cho 1 người dùng. GPOS có thể có nhiều người dùng.
- RTOS tối ưu hóa tài nguyên bộ nhớ.
- Mã của của lõi RTOS có thể mở rộng, có thể chọn lọc các đối tượng của lõi để sử dụng.

2.6.2 Ưu điểm của RTOS

- Lập kế hoạch ưu tiên các tiến trình
  - Các tiến trình quan trọng có thể được xếp lịch để vẫn có thể chạy trong thời gian cố định.
  - Có thể tách biệt tiến trình quan trọng và tiến trình không quan trọng.
- Tính modul, cách dùng stack được xác định
  - RTOS bắt buộc có lõi.
  - Các ngăn xếp, giao thức, phần mềm trung gian đều có thể tùy chỉnh: chúng chỉ xây dựng xung quanh lõi và nên có thể đoán trước mức sử dụng bộ nhớ và kiểm soát các yêu cầu về bộ nhớ.
- Tái sử dụng mã
  - Do RTOS có tính mô-đun nên có thể tái sử dụng mã.

- Sử dụng bộ nhớ hiệu quả
  - RTOS yêu cầu cả bộ nhớ lệnh và RAM để hoạt động nhưng chúng ít dùng hơn so với OS thường.
- Giảm chi phí
  - Khi cài ứng dụng, RTOS có thể yêu cầu thêm code.
  - Nhưng nó vẫn có chi phí thấp hơn và phản hồi nhanh hơn so với giải pháp Polling loop (Vòng lặp bỏ phiếu).
- Chạy đa nền tảng
  - Nhiều RTOS có thể chạy trên nhiều bộ vi điều khiển khác nhau.
- Tính trừu tượng của phần cứng
  - Nhiều RTOS có thể trừu tượng hóa đơn giản phần cứng của các Bộ vi điều khiển giống nhau.

2.6.3 Nhược điểm của RTOS

- Thiếu đa tác vụ
  - Quản lý rất hiệu quả trong việc quản lý lượng nhỏ tác vụ theo lịch trình.
  - Kém hiệu quả khi quản lý đa tác vụ.
- Các tác vụ có mức độ ưu tiên thấp sẽ phải chờ lâu hơn so với OS
- Chỉ có thể chạy đồng thời các tác vụ tối thiểu
- Không thể phân chia bộ nhớ
  - OS phân chia bộ nhớ để tránh xung đột.
  - RTOS thường không có tính năng này.

2.6.4 Bảng so sánh đặc điểm của RTOS và GPOS

Đặc điểm	**RTOS**	**GPOS**
Tiêu chí thiết kế	• Thực hiện chính xác • Thời gian phải đảm bảo	• Thực thi không xác định, không cần đảm bảo về thời gian và thời hạn
Việc sử dụng tài nguyên	• Dùng tối thiểu tài nguyên và quản lý bộ nhớ hiệu quả	• Sử dụng tài nguyên hiệu quả và chú trọng trải nghiệm người dùng
Thuật toán đổi page	• Không cho phép	• Cho phép
Sửa đổi trong chương trình	• Không cho phép	• Cho phép
Chia sẻ tài nguyên với bên ngoài	• Không cho phép	• Cho phép
Xử lý đồng thời nhiều quy trình hoặc ứng dụng	• Chỉ xử lý một	• Có thể xử lý đồng thời nhiều cái
Thời gian phản hồi	• Không hạn chế, thường là vài giây	• Rất ngắn, bị giới hạn nghiêm ngặt
Ưu tiên tác vụ	• Có cơ chế ưu tiên để thực hiện tác vụ quan trọng	• Phải chờ
Ứng dụng	• Các hệ thống cần đảm bảo nghiêm ngặt về thời gian - Ô tô tự lái - Hàng không vũ trụ - Hệ thống điều khiển công nghiệp - Thiết bị y tế - Robot - Radar quân sự	• PC, desktop, smart phone, smart watch, TV
Khi lập trình	• Rất phức tạp • Có phản hồi cụ thể	• Dễ hơn, không rắc rối
Hiệu suất và sự ổn định	• Mang lại hiệu quả cao hơn khi trường hợp giảm tính đồng thời giữa các dịch vụ và chương trình -> Các trường hợp khác thì không	• Hiệu suất ở mức khá và phụ thuộc vào CPU và RAM trong máy

2.7 Phân tích một mô hình thực tế sử dụng RTOS

Cùng với bước tiến của công nghệ, độ nhạy và phản hồi của các cảm biến, hệ thống nhận diện ngày càng được nâng cao. Tuy nhiên, bên cạnh yếu tố về phần cứng, yếu tố phần mềm là hệ điều hành (OS) đôi lúc gặp vấn đề trong việc xử lý các tiến trình/ tác vụ đúng thời gian dự đoán dẫn đến tình trạng chậm tiến trình hoặc tệ hơn là treo máy. Trong những lĩnh vực công việc đặc thù đòi hỏi việc phản hồi lại các tác động từ bên ngoài gần như ngay lập tức, bất cứ sai sót nào về mặt thời gian đều có thể gây ra hậu quả không thể đo lường. Do đó, trong những công việc đặc thù đòi hỏi yếu tố chính xác một cách tuyệt đối và đáp ứng ngay lập tức, RTOS được sử dụng để khắc phục vấn đề trên. Một trong những ví dụ điển hình là áp dụng RTOS trong vận hành thiết bị bay không người lái.

Thiết bị bay không người lái (UAV – Unmanned Aerial Vehicle) là những thiết bị được thiết kế và chế tạo để có thể vận hành từ xa hoặc bán tự động để phục vụ cho những nhiệm vụ, chiến thuật khó có thể can thiệp trực tiếp bằng yếu tố con người. Để việc vận hành UAV được chính xác và hiệu quả, cần một lượng lớn cảm biến và thành phần kỹ thuật phối hợp một cách điều độ và hợp lý. Bất cứ một sai sót nhỏ nào trong quá trình vận hành đều có thể dẫn đến tai nạn và hậu quả nghiêm trọng. Ví dụ như chậm trễ trong việc đối phó với sự xuất hiện đột ngột của các vật cản không được dự đoán trước (chim chóc, cành cây rơi,...) hoặc sự thay đổi đột ngột của hướng gió, cường độ gió đều có thể gây thiệt hại đến cánh quạt, động cơ nếu không được xử lý kịp thời.

Để đảm bảo tính ổn định và tin cậy về khả năng xử lý dữ liệu khách quan từ môi trường trong thời gian cực ngắn, RTOS thường được chọn để sử dụng làm hệ điều hành của các UAV. Để đạt được những yếu tố như trên, nguyên lý vận hành của RTOS trong UAV thường được thiết kế như sau:

- Lập kế hoạch tác vụ (Task Scheduling): RTOS quản lý các tác vụ khác nhau trên UAV như kiểm soát độ cao, hướng bay, xử lý dữ liệu, và giao tiếp. Mỗi tác vụ được ưu tiên dựa trên mức độ quan trọng của nó và thời hạn cụ thể. Hệ thống sử dụng các thuật toán điều độ tiến trình như điều độ quay vòng (Round Robin) hoặc điều độ ưu tiên tần số nhỏ nhất (Rate Monotonic) để quyết định thứ tự thực hiện các tác vụ, đảm bảo rằng các tác vụ quan trọng sẽ không bị trễ.
- Xử lý ngắt (Interrupt Handling): Nhờ RTOS, UAV có khả năng xử lý ngắt từ các cảm biến khi xuất hiện các sự kiện bất ngờ, đảm bảo tính toàn vẹn của vật thể cũng như tái ổn định quá trình bay và thiết lập lại lộ trình mới. RTOS sẽ quản lý việc ưu tiên và xử lý các ngắt này theo độ ưu tiên, đảm bảo rằng các

ngắt quan trọng được xử lý ngay lập tức. Ví dụ trong điều kiện gió lớn, UAV có thể bị ảnh hưởng lộ trình bay, dẫn đến khả năng đâm phải vật cản. Trong trường hợp đó, thông tin phản hồi từ cảm biến siêu âm và hồng ngoại sẽ được ưu tiên trước để xác định vật cản, sau đó sử dụng các cảm biến hướng gió, từ tính để xác định độ cao, vị trí để thiết lập quy trình tránh né và lập lộ trình bay mới.

- Quản lý tài nguyên (Resource Management): RTOS cung cấp quản lý tài nguyên để đảm bảo rằng các tác vụ không xung đột với nhau khi sử dụng tài nguyên chung, chẳng hạn như bộ nhớ, CPU, hoặc các cảm biến. Điều này đảm bảo rằng hệ thống hoạt động một cách ổn định và không gây ra xung đột tài nguyên, luôn có bộ nhớ dư ra để xử lý những tình huống bất ngờ, đảm bảo vận hành trơn tru, tránh tình trạng treo máy dẫn đến hậu quả không lường trước được.
- Xác định (Determinism): Một đặc điểm quan trọng của RTOS là tính xác định trong thời gian hoàn thành các tác vụ. Mỗi tác vụ bay và hiệu chỉnh đều được thiết lập với thời gian hoàn thành cụ thể và không được phép trễ. Điều này đảm bảo rằng UAV có thể thực hiện các tác vụ quan trọng theo đúng thời gian và thực hiện các thao tác thích ứng kịp thời.
- Xử lý lỗi (Error Handling): RTOS cung cấp cơ chế để xử lý lỗi. Nếu một tác vụ gặp lỗi hoặc không thể hoàn thành nhiệm vụ đúng thời hạn đã được định sẵn, RTOS có thể nhanh chóng khôi phục hệ thống và thực hiện các biện pháp khắc phục để đảm bảo an toàn và hoạt động liên tục của UAV. Ví dụ khi một bộ phận cảm biến bị lỗi, thay vì đợi dữ liệu phản hồi từ cảm biến có thể gây ra sai lệch hoặc quá thời gian chờ, RTOS sẽ quyết định sử dụng cảm biến khác và tính toán để bù vào phần tham số bị thiếu. Nếu phần dữ liệu bị thiếu quá lớn, UAV có thể bị cưỡng chế hạ cánh để đảm bảo an toàn.
- Giao tiếp (Communication): RTOS cung cấp các chức năng giao tiếp giữa các tác vụ, cảm biến, và các thành phần khác trong hệ thống UAV. Điều này đảm bảo rằng dữ liệu được truyền đúng cách và đáng tin cậy giữa các thành phần. Đồng thời, RTOS cũng hỗ trợ giao tiếp với các bộ điều khiển mặt đất từ xa, giúp gửi đi dữ liệu nhận được và có thể nhận được hỗ trợ trong điều kiện khắc nghiệt, cần chỉ đạo trực tiếp từ phán đoán và tư duy của người điều khiển.
- Bảo mật (Security): RTOS cần đảm bảo tính bảo mật của hệ thống. Điều này bao gồm quản lý quyền truy cập và mã hóa dữ liệu để ngăn chặn truy cập trái phép hoặc xâm nhập. Rất nhiều UAV không được trang bị tính năng Security khi lập trình viên xây dựng RTOS cho chúng. Điều này dẫn đến việc tin tặc

(hacker) có thể tấn công lớp bảo mật mỏng manh của RTOS, từ đó chiếm quyền điều khiển RTOS. Vấn đề này đặc biệt quan trọng trong lĩnh vực quân sự, khi RTOS có thể được trang bị vũ trang và mang trong mình sứ mệnh quan trọng như hộ tống hoặc giám sát mục tiêu từ xa.

Với RTOS, các bộ phận của UAV có thể nhận được lệnh và hoạt động theo 3 nhiệm vụ chính sau đây:

- Tác vụ điều khiển bay: Tùy thuộc vào loại UAV, tác vụ điều khiển bay có thể được thực hiện bằng cách sử dụng một số thuật toán khác nhau. Một thuật toán phổ biến là thuật toán PID (Proportional Integral Derivative). Thuật toán PID sử dụng ba tham số để điều chỉnh hướng, tốc độ, và độ cao của UAV. RTOS đảm bảo rằng thuật toán PID được thực hiện với tần số chính xác để UAV có thể di chuyển theo cách mong muốn. Ngoài ra, các UAV tiên tiến có thể được thiết lập RTOS tích hợp Mạng khu vực điều khiển thiết bị bay không người lái (UAVCAN – Unmanned Aerial Vehicle Controller Area Network) để có thể giao tiếp liền với trạm kiểm soát mặt đất hoặc bộ điều khiển từ xa mà không cần sự hỗ trợ của Internet.
- Quản lý cảm biến: RTOS được sử dụng để thu thập dữ liệu từ các cảm biến và lưu trữ dữ liệu đó trong bộ nhớ. Dữ liệu này sau đó được sử dụng bởi các tác vụ khác để điều khiển UAV. Nhờ ưu điểm về mặt thời gian, các thông tin từ cảm biến có thể được cập nhật liên tục với độ trễ rất nhỏ (gần như ngay lập tức) giúp các bộ phận khác của UAV có thể ứng phó kịp với các thay đổi khách quan.
- Quản lý động cơ: Nhờ lợi thế thời gian, các dữ liệu từ cảm biến được RTOS tính toán và triển khai tới các động cơ với độ trễ rất nhỏ, từ đó hiệu chỉnh công suất vận hành và làm giảm đáng kể các nguy cơ từ môi trường xung quanh, đồng thời triển khai các phương án đối phó với các trường hợp đặc thù được lập trình viên thiết kế sẵn, giúp UAV có thể ổn định lộ trình bay và tiết kiệm được hao phí nhiên liệu.

CHƯƠNG 3. KẾT LUẬN

Thế kỉ 21, các hệ điều hành đa năng (GPOS) như Windows, Linux,... đã phát triển hơn rất nhiều so với thời điểm chúng ra mắt, nhưng nhờ những cơ chế đặc thù của mình, RTOS vẫn giữ một vai trò quan trọng trong rất nhiều hệ thống nhúng. Gọn nhẹ, chính xác, phản hồi nhanh, RTOS đã trở thành hệ điều hành được các lập trình viên ưu tiên khi tích hợp vào các mô hình nhúng đòi hỏi sự nhỏ gọn và đáng tin cậy. Nhờ có RTOS, rất nhiều sản phẩm tưởng chừng như chỉ có trong công nghệ viễn tưởng đã được hiện thực hóa. Một hệ thống phản hồi gần như ngay tức thì với biến cố, một chiếc vòng đeo tay chỉ có trọng lượng tương đương ba tờ giấy A4 nhưng có khả năng giám sát sức khỏe mọi lúc,... Dần dần, RTOS đã đưa những điều tựa như phép màu đến với thực tại.

Tuy nhiên, RTOS không thể thay thế được GPOS do tính đơn điệu quá lớn. Mặc dù thời gian xử lý nhanh và tối ưu, RTOS lại không thể xử được đa tác vụ và hiển thị thông qua giao diện sử dụng đồ họa như GPOS, gây nên rào cản rất lớn với người dùng cuối bởi không phải thiết bị phần cứng nào cũng yêu cầu độ trễ về phản hồi quá nghiêm ngặt. Bên cạnh đó, tùy vào mục đích và phân cấp ưu tiên của từng tính năng, lập trình viên nên xem xét sự cân bằng giữa các loại RTOS khác nhau, không phải tiêu chuẩn nào cũng có trọng số và ảnh hưởng như nhau. Hơn hết, cho dù công nghệ có phát triển như thế nào, yếu tố con người vẫn là quan trọng nhất. Một công nghệ dù tốt đến đâu nếu không có lập trình viên và đội ngũ phát triển, kiểm thử đầy đủ sẽ không thể đạt được đến sứ mệnh của nó.

TÀI LIỆU THAM KHẢO

[1] *Real Time Operating System in Embedded Systems*. **url:**https://intechhouse.com/blog/real-time-operating-system-im-embedded-systems/ (**urlseen:** 02/11/2023).

[2] Preeti Wadhwani, *Real-time Operating System Market - By Type (Soft, Hard, Firm), By Application (Automotive, Consumer Electronics, Manufacturing, Healthcare, Military & Defence, IT & Telecom) & Forecast 2023 – 2032*. **url:**https://www.gminsights.com/industry-analysis/real-time-operating-system-market (**urlseen:** 02/11/2023).

[3] Shivalibhadaniya, *Fixed-Priority Scheduling*. **url:**https://www.geeksforgeeks.org/fixed-priority-pre-emptive-scheduling/ (**urlseen:** 04/11/2023).

[4] Khoa, *Trạng thái tiến trình*. **url:**https://deviot.vn/blog/rtos-phan-2-task-schedule.60161791 (**urlseen:** 04/11/2023).

[5] Abhisheksharmaabhia, *Round-Robin Algorithm*. **url:**https://www.geeksforgeeks.org/round-robin-scheduling-with-different-arrival-times/ (**urlseen:** 04/11/2023).

[6] *Trạng thái tiến trình*. **url:**https://aticleworld.com/rtos-task-states/ (**urlseen:** 04/11/2023).

[7] Ajaychawla, *Rate-Monotonic Scheduling*. **url:**https://www.geeksforgeeks.org/rate-monotonic-scheduling/ (**urlseen:** 04/11/2023).

[8] Quàng Hoàng Anh, *Trạng thái tiến trình*. **url:**https://epcb.vn/blogs/news/hoat-dong-cua-he-dieu-hanh-thoi-gian-thuc-rtos (**urlseen:** 04/11/2023).

[9] Kondalalith, *Deadline Monotonic*. **url:**https://www.geeksforgeeks.org/difference-between-rate-monotonic-and-deadline-monotonic-scheduling/ (**urlseen:** 04/11/2023).

[10] ShivamKumar, *Earliest-Deadline First – EDF*. **url:**https://www.geeksforgeeks.org/earliest-deadline-first-edf-cpu-scheduling-algorithm/ (**urlseen:** 04/11/2023).

[11] MKS075, *Difference between Time Sharing OS and Real-Time OS*.**url:**https://www.geeksforgeeks.org/difference-between-time-sharing-os-and-real-time-os/ (**urlseen:** 11/11/2023).

[12] Md. Sajid, *Difference Between RTOS and OS*. **url:**https://www.tutorialspoint.com/difference-between-rtos-and-os (**urlseen:** 11/11/2023).

[13] JavaTpoint, *Difference between Real-Time operating system and general-purpose operating system*. **url:**https://www.javatpoint.com/real-time-operating-system-vs-general-purpose-operating-system (**urlseen:** 11/11/2023).

[14] Colin Walls, *Other RTOS services*. **url:**https://www.embedded.com/other-rtos-services/ (**urlseen:** 11/11/2023).

[15] Ian Ferguson, *WHAT ARE THE MOST POPULAR REAL-TIME OPERATING SYSTEMS?*. **url:**https://www.lynx.com/embedded-systems-learning-center/most-popular-real-time-operating-systems-rtos (**urlseen:** 11/11/2023).

[16] Vatsalkumar H. Shah, Dr. Apurva Shah, "An Analysis and Review on Memory Management Algorithms for Real Time Operating System", *International Journal of Computer Science and Information Security*, vol. 14, no. 5, 2016.

[17] Dipti Diwase, Shraddha Shah, Tushar Diwase, Priya Rathod, "Survey Report on Memory Allocation Strategies for Real Time Operating System in Context with Embedded Devices", *International Journal of Engineering*, Vol. 2, Issue 3, pp.1151-1156, 2012.

[18] Shar Vatsalkumar Hasmukhabhai, "Memory Management in Real-Time Operating System", Ph.D. dissertation, The Maharada Sayajirao University Of Baroda, Vadodara - India, 2018.

[19] Niranjan Ravi, Mohamed El-Sharkawy, "Integration of UAVs with Real Time Operating Systems using UAVCAN", 2019 IEEE 10th Annual Ubiquitous Computing - Electronics & Mobile Communication Conference, pp.0600–0605, 2019.

[20] Geoffrey Hunter, *A Comparison Of Serialization Formats*. **url:**https://blog.mbedded.ninja/programming/serialization-formats/a-comparison-of-serialization-formats/ (**urlseen:** 11/11/2023).

[21] Himanshu Arora, *File Handling in C with Examples (fopen, fread, fwrite, fseek)*. **url:**https://www.thegeekstuff.com/2012/07/c-file-handling/ (**urlseen:** 11/11/2023).

[22] The Department of Computer Science at the University of Chicago, *FILE*

I/O. **url:**https://www.classes.cs.uchicago.edu/archive/2017/winter/51081-1/LabFAQ/lab2/fileio.html (**urlseen:** 11/11/2023).

[23] Tanya Bahrynovska, *Exploring the Basics of Real-Time Operating Systems (RTOS)*. **url:**https://forbytes.com/blog/basics-of-real-time-operating-system-rtos/ (**urlseen:** 02/11/2023).

PHÂN CHIA CÔNG VIỆC

Mã sinh viên	Họ và tên	Đóng góp
B21DCCN398	Vũ Huy Hoàng (Thư kí)	• Chương 1. Giới thiệu chung về RTOS • Viết báo cáo (17%)
B21DCCN517	Nguyễn Văn Mạnh	• 2.1. Quản lí File • 2.2. Quản lí bộ nhớ (14%)
B21DCCN546	Lê Đoàn Ngọc Nam	• 2.2. Quản lí bộ nhớ (10%)
B21DCAT062	Nguyễn Thế Độ	• 2.3. Quản lí tiến trình (13%)
B21DCAT186	Nguyễn Văn Trí	• 2.4. Các loại dịch vụ cung cấp bởi RTOS • 2.5. Các biến thể của RTOS (13%)
B21DCCN221	Trần Hữu Đạt	• 2.6. So sánh RTOS và OS (13%)
B21DCAT089	Nguyễn Đức Hiếu (Nhóm trưởng)	• 2.7. Phân tích một mô hình thực tế sử dụng RTOS • 2.2. Quản lí bộ nhớ • Chương 3. Kết luận • Làm slide (20%)