Mục lục

[DANH MỤC HÌNH VẼ 3](#_Toc533098439)

[MỞ ĐẦU 4](#_Toc533098440)

[PHẦN I: NGUYÊN LÍ HỆ ĐIỀU HÀNH 5](#_Toc533098441)

[CHƯƠNG 1: CƠ SỞ LÍ THUYẾT 5](#_Toc533098442)

[1. Hệ điều hành thời gian thực 5](#_Toc533098443)

[2. Lập lịch thời gian thực 6](#_Toc533098444)

[3. Một số thuật toán lập lịch thời gian thực 9](#_Toc533098445)

[ Thuật toán Deadline Monotonic 9](#_Toc533098446)

[ Thuật toán Rate Monotonic 10](#_Toc533098447)

[ Thuật toán Earlier Deadline First 11](#_Toc533098448)

[CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG 12](#_Toc533098449)

[1. Yêu cầu 12](#_Toc533098450)

[2. Phân tích và thiết kế hệ thống 12](#_Toc533098451)

[ Sơ đồ hoạt động của các thuật toán: 12](#_Toc533098452)

[ Môi trường triển khai 13](#_Toc533098453)

[CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 13](#_Toc533098454)

[1. Cài đặt chương trình 13](#_Toc533098455)

[ Lớp Task 13](#_Toc533098456)

[ Nhập danh sách tác vụ 14](#_Toc533098457)

[ Tính CPU Utilization 14](#_Toc533098458)

[ Tính Hyper Period 15](#_Toc533098459)

[ Khởi tạo danh sách các tác vụ sẵn sàng theo độ ưu tiên 15](#_Toc533098460)

[ Deadline Monotonic 15](#_Toc533098461)

[ Rate Monotonic 15](#_Toc533098462)

[ Earlier Deadline First 16](#_Toc533098463)

[ Xử lí lập lịch 16](#_Toc533098464)

[2. Kết quả thực hiện 17](#_Toc533098465)

[3. Đánh giá kết quả 18](#_Toc533098466)

[KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 18](#_Toc533098467)

[PHẦN II: LẬP TRÌNH MẠNG 19](#_Toc533098468)

[CHƯƠNG 1: CƠ SỞ LÍ THUYẾT 19](#_Toc533098469)

[1. IP Helper 19](#_Toc533098470)

[ Tổng quát về IP Helper 19](#_Toc533098471)

[ Mục đích của IP Helper 20](#_Toc533098472)

[ Nơi áp dụng IP Helper 20](#_Toc533098473)

[ Đối tượng phát triển 20](#_Toc533098474)

[ Các yêu cầu về môi trường run-time 20](#_Toc533098475)

[2. Windows Sockets 2 20](#_Toc533098476)

[ Mục đích 20](#_Toc533098477)

[ Đối tượng phát triển 21](#_Toc533098478)

[ Các yêu cầu về môi trường run-time 21](#_Toc533098479)

[3. Sử dụng IP Helper 21](#_Toc533098480)

[4. Lệnh netstat 23](#_Toc533098481)

[ Cú pháp 23](#_Toc533098482)

[ Tham chiếu 23](#_Toc533098483)

[ Chú giải 24](#_Toc533098484)

[CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG 25](#_Toc533098485)

[1. Yêu cầu 25](#_Toc533098486)

[2. Phương pháp triển khai 25](#_Toc533098487)

[CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 29](#_Toc533098488)

[1. Cài đặt chương trình 29](#_Toc533098489)

[ Hàm lấy thông tin các kết nối TCP đang hoạt động 29](#_Toc533098490)

[ Hàm lấy thông tin các cổng UDP trên đó máy tính đang lắng nghe 31](#_Toc533098491)

[ Hàm lấy thông tin thống kê theo giao thức 32](#_Toc533098492)

[ Chương trình chính 36](#_Toc533098493)

[2. Kết quả chạy chương trình 37](#_Toc533098494)

[3. Đánh giá kết quả 38](#_Toc533098495)

[KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 38](#_Toc533098496)

DANH MỤC HÌNH VẼ

[Hình 1: Các trạng thái của một tác vụ 7](#_Toc533098499)

[Hình 2: Ví dụ cho thuật toán Deadline Monotonic 10](#_Toc533098500)

[Hình 3: Ví dụ về thuật toán Rate Monotonic 11](#_Toc533098501)

[Hình 4: Ví dụ về thuật toán Earlier Deadline First 12](#_Toc533098502)

[Hình 5: Sơ đồ hoạt động của các thuật toán 13](#_Toc533098503)

[Hình 6: Kết quả chương trình Deadline Monotonic 17](#_Toc533098504)

[Hình 7: Kết quả chương trình Rate Monotonic 17](#_Toc533098505)

[Hình 8: Kết quả chương trình Earlier Deadline First 18](#_Toc533098506)

[Hình 9: Kết quả chạy chương trình với tùy chọn my\_netstat 37](#_Toc533098507)

[Hình 10: Kết quả chạy chương trình với tùy chọn my\_netstat –p udp -a 37](#_Toc533098508)

[Hình 11: Kết quả chạy chương trình với tùy chọn my\_netstat -s 38](#_Toc533098509)

DANH SÁCH BẢNG BIỂU

[Bảng 1: Danh sách tham chiếu của lệnh netstat 11](#_Toc532855042)

[Bảng 2: Danh sách tùy chọn của chương trình my\_netstat 12](#_Toc532855043)

DANH SÁCH TỪ VIẾT TẮT

MỞ ĐẦU

PHẦN I: NGUYÊN LÍ HỆ ĐIỀU HÀNH

TIÊU ĐỀ: Tìm hiểu và mô phỏng các giải thuật lập lịch trên hệ thống thời gian thực

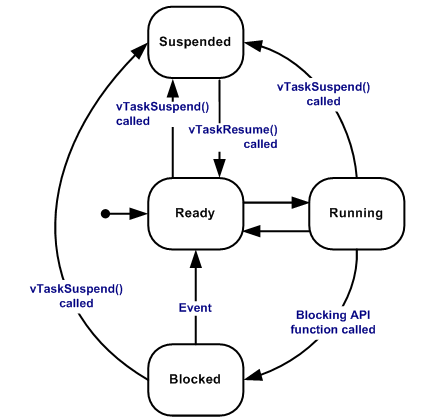
CHƯƠNG 1: CƠ SỞ LÍ THUYẾT

1. Hệ điều hành thời gian thực

* Một hệ điều hành thời gian thực (real-time operating system (RTOS)) là một hệ điều hành (OS) nhằm phục vụ các ứng dụng thời gian thực, xử lý dữ liệu khi nó đi vào, mà không có sự chậm trễ của bộ đệm. Các yêu cầu thời gian xử lý (bao gồm bất kỳ sự chậm trễ nào của hệ điều hành) được tính bằng phần mười của giây hoặc thời gian ngắn hơn. Chúng hoặc là được sự kiện điều khiển hoặc chia sẻ thời gian. Các hệ thống do sự kiện điều khiển chuyển đổi giữa các nhiệm vụ dựa trên các ưu tiên của chúng trong khi các hệ thống chia sẻ thời gian chuyển công việc dựa trên ngắt đồng hồ.
* Một đặc điểm chính của một RTOS là mức độ nhất quán liên quan đến số lượng thời gian cần để chấp nhận và hoàn thành nhiệm vụ của một ứng dụng; sự thay đổi là rung pha (jitter: những biến đổi về pha của tín hiệu số thu được so với vị trí lí tưởng của chúng có tần số lớn hơn hoặc bằng 10 Hz).Một hệ điều hành thời gian thực "cứng" có ít jitter hơn một hệ điều hành thời gian thực "mềm". Mục tiêu thiết kế chính không phải là thông lượng cao, mà là sự bảo đảm về một loại hiệu suất mềm hoặc cứng. Một RTOS thường hoặc nói chung đáp ứng được một thời hạn quy định là một hệ điều hành thời gian thực mềm, nhưng nếu nó có thể đáp ứng được một thời hạn xác định thì đó là một hệ điều hành thời gian thực cứng.
* Một RTOS có một toán tiên tiến cho việc lập thời biểu (scheduling). Tính linh hoạt của bộ lập thời biểu cho phép dàn xếp hệ thống máy tính rộng rãi hơn về các ưu tiên quá trình, nhưng hệ điều hành thời gian thực thường xuyên hơn dành cho một bộ nhỏ các ứng dụng. Các yếu tố chính trong một hệ điều hành thời gian thực là độ trễ ngắt tối thiểu và độ trễ chuyển luồng (thread) tối thiểu; một hệ điều hành thời gian thực được đánh giá cao hơn về mức độ nhanh nhạy hoặc có thể dự đoán được nó như thế nào so với số lượng công việc nó có thể thực hiện trong một khoảng thời gian nhất định.

1. Lập lịch thời gian thực

* Một tác vụ thường có các trạng thái:
* Chạy (Running)
* Khi một tác vụ đang thực sự thực thi, nó được gọi là ở trạng thái Chạy. Nó hiện đang sử dụng bộ xử lý. Nếu bộ xử lý mà RTOS đang chạy chỉ có một lõi thì chỉ có thể có một tác vụ ở trạng thái Chạy tại bất kỳ thời điểm nào.
* Sẵn sàng (Ready)
* Các tác vụ đã sẵn sàng là những tác vụ có thể thực thi (chúng không ở trạng thái Bị chặn hoặc Bị treo) nhưng hiện không thực thi vì một tác vụ khác có mức độ ưu tiên cao hơn hoặc bằng nhau đã ở trạng thái Chạy.
* Bị chặn (Blocked)
* Một tác vụ được cho là ở trạng thái Bị chặn nếu hiện tại nó đang chờ một sự kiện tạm thời hoặc bên ngoài. Ví dụ: nếu một tác vụ gọi vTaskDelay () thì nó sẽ chặn (được đặt ở trạng thái Bị chặn) cho đến khi hết thời gian trì hoãn - một sự kiện tạm thời. Nhiệm vụ cũng có thể chặn để chờ hàng đợi, semaphore, nhóm sự kiện, thông báo hoặc sự kiện semaphore. Các tác vụ ở trạng thái Bị chặn thường có thời gian 'hết thời gian', sau đó tác vụ sẽ hết thời gian và được bỏ chặn, ngay cả khi sự kiện mà tác vụ đang chờ không xảy ra.
* Các tác vụ ở trạng thái Bị chặn không sử dụng bất kỳ thời gian xử lý nào và không thể được chọn để vào trạng thái Chạy.
* Bị treo (Suspended)
* Giống như các tác vụ ở trạng thái Bị chặn, các tác vụ ở trạng thái Bị treo có thể được chọn để vào trạng thái Chạy, nhưng các tác vụ ở trạng thái Tạm dừng không có thời gian chờ. Thay vào đó, các tác vụ chỉ nhập hoặc thoát trạng thái Tạm dừng khi được lệnh rõ ràng thực hiện thông qua các lệnh gọi API vTaskSuspend () và xTaskResume () tương ứng.



Hình 1: Các trạng thái của một tác vụ

* Mô hình tác vụ điển hình cho hệ thống thời gian thực thường bao gồm các thông số:
* Thời hạn của tác vụ (Deadline): D
* Khoảng thời gian của tác vụ (Period): P
* Thời gian thực thi trong trường họp xấu nhất của tác vụ (WCET): C
* Sự dùng CPU (CPU Utilization):
* CPU Utilization của một tác vụ:
* CPU Utilization của tập hợp các tác vụ: ,

Trong đó n là số lượng tác vụ, là thời gian thực thi trong trường hợp xấu nhất của tác vụ thứ i, là khoảng thời gian của tác vụ thứ i.

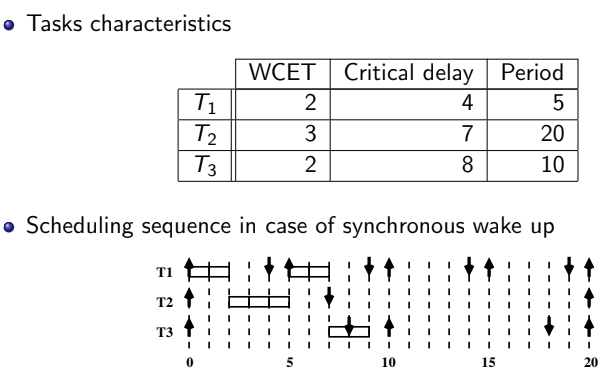
* CPU Utilization là thước đo sự bận rộn của bộ xử lí có thể lặp lại trong chu kì ngắn nhất:
* U > 1 (Quá tải – Overload): một số tác vụ sẽ không đáp ứng được thời hạn.
* U <= 1: Sẽ phụ thuộc và các thuật toán lập lịch. Nếu U = 1 và CPU luôn bận, tất cả thời hạn sẽ được đáp ứng.
* Hầu hết các tác vụ đều bị chặn hoặc sẵn sàng hầu hết thời gian vì nhìn chung chỉ có một tác vụ có thể chạy tại một thời điểm trên mỗi CPU. Số lượng các mục trong hàng đợi sẵn sàng có thể khác nhau rất nhiều, tùy thuộc vào số lượng tác vụ mà hệ thống cần thực hiện và loại lịch trình mà hệ thống sử dụng. Trên các hệ thống không đa nhiệm đơn giản nhưng vẫn đa nhiệm, một tác vụ phải bỏ thời gian trên CPU cho các tác vụ khác, điều này có thể khiến hàng đợi sẵn sàng có số lượng tác vụ tổng thể lớn hơn trong trạng thái sẵn sàng để thực thi (bỏ đói tài nguyên) .
* Thông thường, cấu trúc dữ liệu của danh sách sẵn sàng trong bộ lập lịch được thiết kế để giảm thiểu thời gian trong trường hợp xấu nhất trong phần quan trọng của bộ lập lịch, trong đó một số trường hợp bị cấm, và trong một số trường hợp, tất cả các ngắt đều bị tắt. Nhưng việc lựa chọn cấu trúc dữ liệu cũng phụ thuộc vào số lượng tác vụ tối đa có thể có trong danh sách sẵn sàng.
* Nếu không bao giờ có nhiều hơn một vài nhiệm vụ trong danh sách sẵn sàng, thì một danh sách các nhiệm vụ sẵn sàng được liên kết đôi có khả năng là tối ưu. Nếu danh sách sẵn sàng thường chỉ chứa một vài nhiệm vụ nhưng đôi khi chứa nhiều hơn, thì danh sách nên được sắp xếp theo mức độ ưu tiên. Theo cách đó, việc tìm kiếm nhiệm vụ ưu tiên cao nhất để chạy không yêu cầu lặp qua toàn bộ danh sách. Chèn một nhiệm vụ sau đó yêu cầu đi bộ danh sách sẵn sàng cho đến khi đến cuối danh sách hoặc một nhiệm vụ có mức độ ưu tiên thấp hơn nhiệm vụ được chèn.
* Chăm sóc phải được thực hiện để không ức chế sự ưu tiên trong tìm kiếm này. Các phần quan trọng dài hơn nên được chia thành các phần nhỏ. Nếu xảy ra gián đoạn làm cho một nhiệm vụ ưu tiên cao sẵn sàng trong khi chèn một nhiệm vụ ưu tiên thấp, thì nhiệm vụ ưu tiên cao đó có thể được chèn và chạy ngay lập tức trước khi chèn nhiệm vụ ưu tiên thấp.
* Thời gian phản hồi quan trọng, đôi khi được gọi là thời gian bay lại, là thời gian cần thiết để xếp hàng một tác vụ sẵn sàng mới và khôi phục trạng thái của nhiệm vụ ưu tiên cao nhất để chạy. Trong một RTOS được thiết kế tốt, việc sẵn sàng thực hiện một nhiệm vụ mới sẽ mất từ ​​3 đến 20 hướng dẫn cho mỗi mục nhập hàng đợi sẵn sàng và việc khôi phục nhiệm vụ sẵn sàng ưu tiên cao nhất sẽ mất từ ​​5 đến 30 hướng dẫn.
* Trong các hệ thống tiên tiến hơn, các tác vụ thời gian thực chia sẻ tài nguyên tính toán với nhiều tác vụ không theo thời gian thực và danh sách sẵn sàng có thể dài tùy ý. Trong các hệ thống như vậy, một danh sách sẵn sàng lên lịch được thực hiện như một danh sách được liên kết sẽ không đầy đủ.

1. Một số thuật toán lập lịch thời gian thực

* Thuật toán Deadline Monotonic
* Deadline Monotonic là một chính sách lập lịch ưu tiên với độ ưu tiên cố định.
* Với Deadline Monotonic, các tác vụ (Task) được phân công ưu tiên dựa vào thời hạn (Deadline time). Tác vụ có thời hạn ngắn nhất được chỉ định mức ưu tiên cao nhất. Chính sách lập lịch này sẽ tối ưu cho tập hợp các tác vụ định kỳ (Periodic) hoặc lẻ tẻ (Sporadic) tuân theo các hạn chế sau:
* Tất cả các tác vụ phải có thời hạn (Deadline) nhỏ hơn hoặc bằng thời gian tối thiểu giữa các lần đến của chúng (Period).
* Tất cả các tác vụ có thời gian thực thi (WCET) trong trường hợp xấu nhất nhỏ hơn hoặc bằng thời hạn của chúng.
* Tất cả các tác vụ đều độc lập và không ngăn chặn thực thi của nhau.
* Không có tác vụ nào tự treo.
* Có một số thời điểm, xem như là thời điểm quan trọng, tại đó tất cả tác vụ có thể sẵn sang thực thi đồng thời.
* Chi phí lập lịch (Chuyển từ tác vụ này đến tác vụ khác) bằng 0.
* Tất cả các tác vụ đều có jitter phát hành (thời gian khi các tác vụ đến nó trở nên sẵn sàng để thực thi) bằng 0.
* Deadline Monotonic sẽ không tối ưu cho lập lịch ưu tiên với độ ưu tiên không cố định.
* Kiểm ràng buộc sử dụng cho thuật toán Deadline Monotonic

,

Trong đó n là số lượng tác vụ, là thời gian thực thi trong trường hợp xấu nhất của tác vụ thứ i, là thời hạn của tác vụ thứ i.

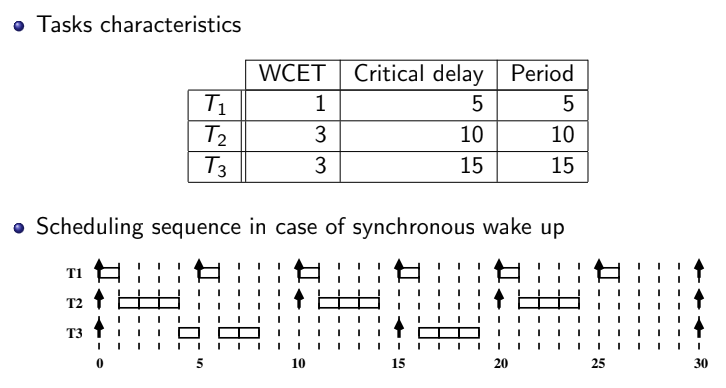


Hình 2: Ví dụ cho thuật toán Deadline Monotonic

* Thuật toán Rate Monotonic
* Trong khoa học máy tính, Rate Monotonic là một thuật toán lập lịch gán sự ưu tiên được sử dụng trong cái hệ điều hành thời gian thực (RTOS). Các ưu tiên tĩnh được xác định dựa vào khoảng thời gian của tác vụ, tác vụ có khoảng thời gian ngắn hơn thì độ ưu tiên công việc sẽ cao hơn.
* Một phiên bản đơn giản của thuật toán Rate Monotonic giả định rằng các tác vụ có các thuộc tính sau:
* Không có sự chia sẽ tài nguyên (ví dụ: tài nguyên phần cứng, hàng đợi,...).
* Thời hạn của tác vụ bằng đúng khoảng thời gian của tác vụ đó (D = P).
* Độ ưu tiên tĩnh (tác vụ có độ ưu tiên tĩnh cao nhất có thể chạy được ngay lập tức trước tất cả các tác vụ khác).
* Các tác vụ có khoảng thời gian / thời hạn ngắn hơn có độ ưu tiên cao hơn.
* Kiểm tra ràng buộc sử dụng cho thuật toán RMS:

,

Trong đó n là số lượng tác vụ, là thời gian thực thi trong trường hợp xấu nhất của tác vụ thứ i, là khoảng thời gian của tác vụ thứ i.

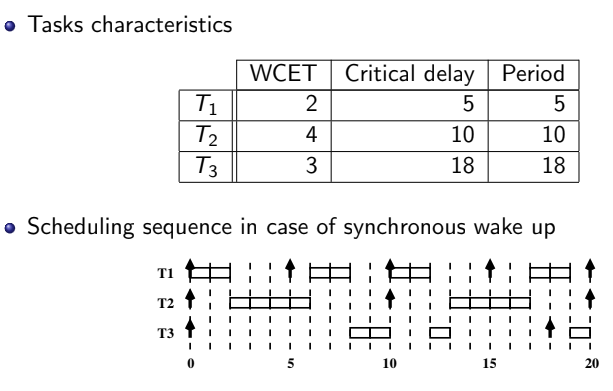


Hình 3: Ví dụ về thuật toán Rate Monotonic

* Thuật toán Earlier Deadline First
* Thuật toán Earlier Deadline First là một thuật toán lập lịch ưu tiên động được sử dụng trong các hệ điều hành thời gian thực để đặt các tiến trình trong hàng đợi ưu tiên. Bất kì khi nào một sự kiện lập lịch xảy ra (một tác vụ kết thúc, tác vụ mới được phát hành,…), hàng đợi sẽ tìm kiếm tác vụ gần nhất với thời hạn (Deadline) của nó. Quá trình này tiếp tục để lên lịch được thực hiện.
* EDF là một thuật toán lập lịch tối ưu trên các bộ xử lý ưu tiên, theo nghĩa sau: nếu một tập hợp các tác vụ độc lập, mỗi tác vụ được đặc trưng bởi thời gian đến, yêu cầu thực hiện và thời hạn, có thể được lên lịch (theo bất kỳ thuật toán nào) theo cách đảm bảo tất cả các tác vụ hoàn thành theo thời hạn của chúng, EDF sẽ lên lịch cho tập hợp các tác vụ này để tất cả chúng hoàn thành theo thời hạn.
* Với việc lên lịch các quy trình định kỳ có thời hạn bằng với thời gian của họ, EDF có mức sử dụng là 100%.
* Kiểm tra rang buộc sử dụng cho thuật toán Earlier Deadline First:

,

Trong đó n là số lượng tác vụ, là thời gian thực thi trong trường hợp xấu nhất của tác vụ thứ i, là khoảng thời gian của tác vụ thứ i.



Hình 4: Ví dụ về thuật toán Earlier Deadline First

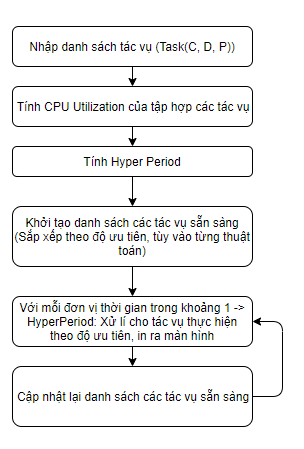
CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

1. Yêu cầu

* Viết chương trình mô phỏng các giải thuật lập lịch trên hệ điều hành thời gian thực
* Thuật toán Deadline Monotonic
* Thuật toán Rate Monotonic
* Thuật toán Earlier Deadline First

1. Phân tích và thiết kế hệ thống

* Sơ đồ hoạt động của các thuật toán:



Hình 5: Sơ đồ hoạt động của các thuật toán

* Môi trường triển khai
* Ngôn ngữ lập trình: C#.
* Công cụ lập trình: Visial Studio 2017 Community.

CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

1. Cài đặt chương trình

* Lớp Task



* Nhập danh sách tác vụ
* Khởi tạo danh sách tác vụ



* Nhập dánh sách tác vụ



* Tính CPU Utilization



* Tính Hyper Period



* Khởi tạo danh sách các tác vụ sẵn sàng theo độ ưu tiên
* Deadline Monotonic



* Rate Monotonic



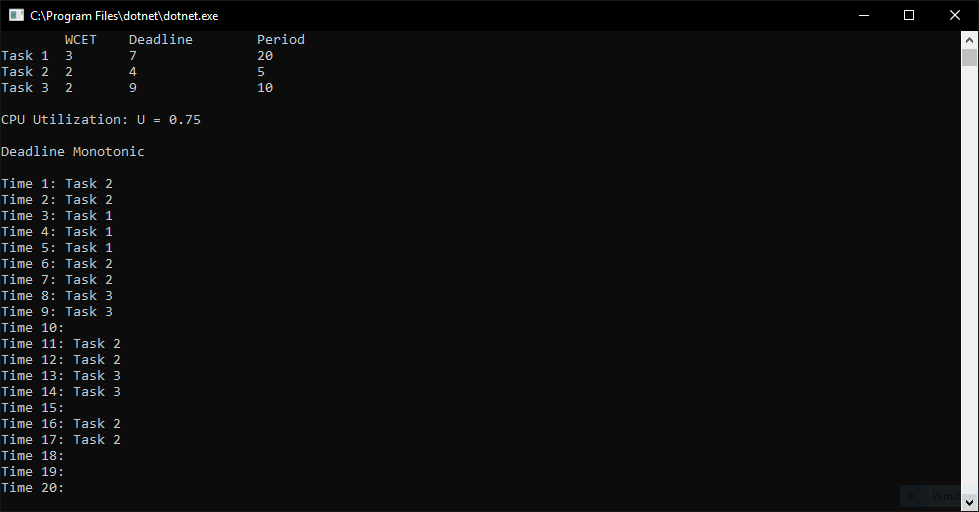
* Earlier Deadline First



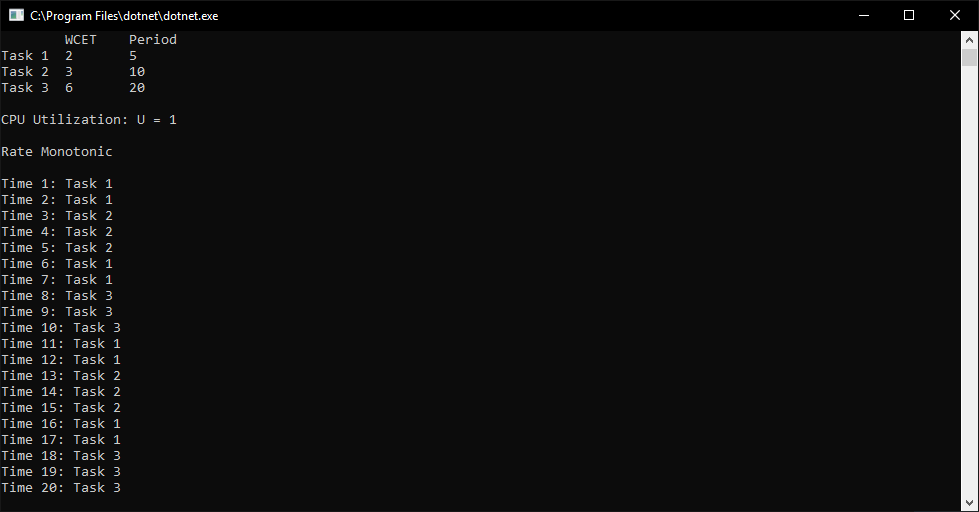
* Xử lí lập lịch



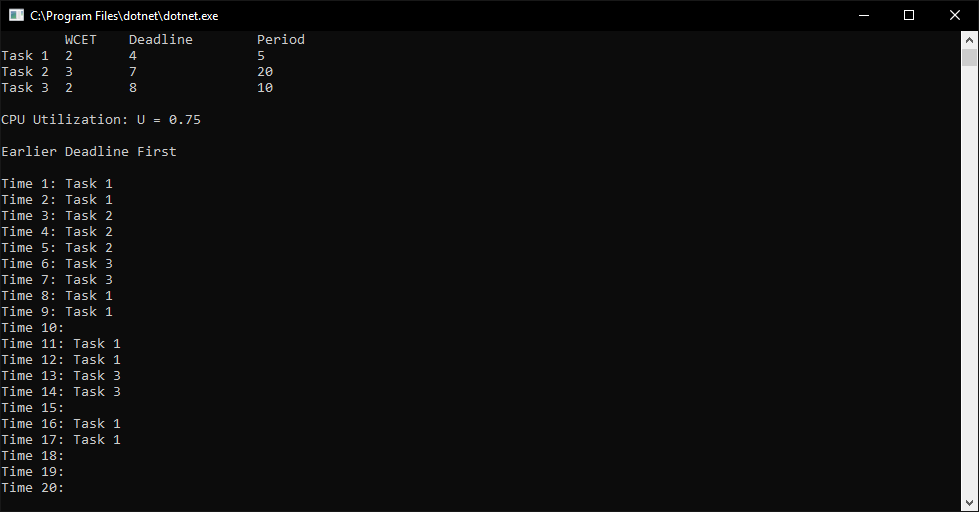
1. Kết quả thực hiện



Hình 6: Kết quả chương trình Deadline Monotonic



Hình 7: Kết quả chương trình Rate Monotonic



Hình 8: Kết quả chương trình Earlier Deadline First

1. Đánh giá kết quả

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

PHẦN II: LẬP TRÌNH MẠNG

TIÊU ĐỀ: Tìm hiểu về IP Helper và xây dựng chương trình my\_netstat

Yêu cầu đề tài:

1. Tìm hiểu về IP Helper và chương trình netstat.
2. Xây dựng được chương trình my\_netstat để mô phỏng chương trình netstat.

CHƯƠNG 1: CƠ SỞ LÍ THUYẾT

1. IP Helper

* Tổng quát về IP Helper
* Trình trợ giúp giao thức Interenet (IP Helper) hỗ trợ quản trị mạng cho máy tính cục bộ bằng cách cho phép các ứng dụng truy xuất thông tin về cấu hình của mạng của máy tính cục bộ và sửa đổi cấu hình đó. IP Helper cũng cung cấp các cơ chế thông báo để đảm bảo rằng một ứng dụng được thông báo khi các khía cạnh nhất định của cấu hình mạng của máy tính cục bộ bị thay đổi.
* Nhiều hàm của IP Helper truyền các tham số có cấu trúc đại diện cho các loại dữ liệu được liên kết với công nghệ MIB (Management Information Base – là một cấu trúc dữ liệu gồm các đối tượng được quản lý, được dùng cho các thiết bị chạy trên nền tảng TCP/IP). IP Helper sử dụng các cấu trúc này để thể hiện các thông tin mạng khác nhau, chẳng hạn như các mục đệm ARP (Address Resolution Protocol – giao thức phân giải địa chỉ). Mặc dù IP Helper API sử dụng các cấu trúc cấu trúc này nhưng IP Helper khác biệt so với MIB và SNMP (Simple Network Management Protocol – quản lí giao thức mạng đơn giản).
* IP Helper cung cấp các khả năng áp dụng trong các lĩnh vực sau:
* Lấy thông tin cấu hình mạng
* Quản lí bộ điều hợp mạng
* Quản lí giao diện
* Quản lí các địa chỉ IP
* Sử dụng giao thức phân giải địa chỉ
* Lấy thông tin về giao thức Internet và giao thức tin nhắn điều khiển Internet
* Quản lí định tuyến
* Nhận thông báo về các sự kiện mạng
* Lấy thông tin về giao thức điều khiển truyền vận (TCP) và giao thức dữ liệu người dùng (UDP).
* Mục đích của IP Helper
* IP Helper API cho phép truy xuất và sửa đổi cài đặt cấu hình mạng cho máy tính cục bộ.
* Nơi áp dụng IP Helper
* IP Helper API có thể được áp dụng trong bất kì môi trường điện toán nào có cấu hình mạng và cấu hình TCP/IP có thể lập trình được. Một số ứng dụng điển hình bao gồm các giao thức định tuyến IP và các chương trình SNMP (Simple Network Management Protocol – quản lí giao thức mạng đơn giản).
* Đối tượng phát triển
* IP Helper API được thiết kế để sử dụng bởi các nhà lập trình viên C/C++.
* Các yêu cầu về môi trường run-time
* IP Helper API có thể được sử dụng trên tất cả các nền tảng Windows. Không phải tất cả hệ điều hành đều hỗ trợ tất cả các hàm của IP Helper. Nếu một hàm của IP Helper được sử dụng trên một nền tảng không được hỗ trợ, lỗi ERROR\_NOT\_SUPPORTED sẽ được trả về.

1. Windows Sockets 2

* Mục đích
  + Windows Sockets 2 (Winsock) cho phép các nhà lập trình viên tạo ra các ứng dụng internet, intranet hoặc có thể kết nối mạng nâng cao để truyền dữ liệu ứng dụng qua mạng, độc lập với giao thức mạng đang sử dụng. Với Winsock, lập trình viên được cung cấp khả nâng kết nối mạng nâng cao của hệ điều hành Microsoft Windows chẳng hạn như multicast (phát đa hướng) và Quality of Service (QoS – chất lượng của dịch vụ).
* Winsock theo mô hình Windows Open System Architecture (kiến trúc hệ thống mở Windows), nó xác định giao diện nhà cung cấp dịch vụ tiêu chuẩn (SPI) giữa giao diện lập trình ứng dụng (API) với với các chức năng được xuất của nó và các ngăn xếp giao thức. Winsock sử dụng mô hình socket được phổ biến lần đầu tiên bởi Berkeley Software Distribution (BSD) UNIX. Sau đó, nó đã được điều chỉnh cho Windows trong Windows Sockets 1.1, trong đó các ứng dụng Windows Sockets 2 tương thích ngược. Lập trình Winsock trước đây tập trung vào TCP / IP. Một số thực tiễn lập trình hoạt động với TCP / IP không hoạt động với mọi giao thức. Do đó, API Windows Sockets 2 thêm các chức năng khi cần thiết để xử lý một số giao thức.
* Đối tượng phát triển
  + Windows Sockets 2 được thiết kế để sử dụng bởi các nhà lập trình viên C/C++. Làm quen với mạng Windows là bắt buộc.
* Các yêu cầu về môi trường run-time
  + Windows Sockets 2 có thể sử dụng trên tất cả các nền tảng Windows.

1. Sử dụng IP Helper

* Dưới đây là các bước để tạo viết một ứng dụng sử dụng IP Helper
* Tạo mới một project C++ và thêm mới một source file C++ vào project.
* Đảm bảo rằng Build environment đã đề cập đến các thư mục Include, Lib và Src của Platform Software Development Kit (SDK).
* Đảm bảo rang Buil environment đã liên kết đến file IP Helper Library Iphlpapi.lib và Winsock Library WS2\_32.lib.
* Bắt đầu lập trình ứng dụng IP Helper. Sử dụng IP Helper API bằng cách include các IP Helper header file vào source file.



* Một số lưu ý:
  + Tệp tiêu đề Iphlpapi.h (header file) là bắt buộc đối với các ứng dụng sử dụng IP Helper. Iphlpapi.h sẽ tự động bao gồm các tệp tiêu đề khác với các cấu trúc (strutures) và các bảng liệt kê (enumerations) được sử dụng bởi IP Helper.
  + Tệp tiêu đề Winsock2.h cho Windows Sockets 2 được yêu cầu bởi hầu hết các ứng dụng sử dụng IP Helper. Dòng #include <winsock2.h> phải được đặt trước dòng #include <iphlpapi.h>
  + Tệp tiêu đề winsock2.h bên trong bao gồm các thành phần chính từ tệp windows.h, do đó thường không có dòng #include <windows.h> trong chương trình sử dụng IP Helper. Nếu cần thiết có thể dùng dòng lệnh trên nhưng phải bắt đầu bằng #define WIN32\_LEAN\_AND\_MEAN macro. Vì các lí do lịch sử, windows.h mặc định đã bao gồm winsock.h cho Windows Sockets 1.1. Các khai báo trong winsock2.h sẽ xung đột với các khai báo trong windows.h. Macro WIN32\_LEAN\_AND\_MEAN sẽ ngăn chặn việc tệp winsock.h được bao gồm bởi tệp windows.h.

1. Lệnh netstat

Lệnh netstat có chức năng hiển thị các kết nối TCP đang hoạt động, các cổng mà tại đó máy tính đang nghe và đồng thời hiển thị các thống kê Ethernet, bảng định tuyến IP, thống kê IPv4 (cho giao thức IP, ICMP, TCP và UDP) và số liệu thống kê IPv6 (cho IPv6, ICMPv6, TCP qua IPv6 và UDP qua giao thức IPv6). Nếu không có tham số, netstat sẽ hiển thị các kết nối TCP đang hoạt động.

* Cú pháp



* Tham chiếu

|  |  |
| --- | --- |
| Tham biến | Mô tả |
| -a | Hiển thị tất cả các kết nối TCP đang hoạt động, cũng như các cổng TCP và UDP mà trên đó máy tính đang nghe. |
| -e | Hiển thị số liệu thống kê Ethernet, chẳng hạn như số byte và các gói được gửi và nhận. Tham số này có thể được kết hợp với -s. |
| -n | Hiển thị các kết nối TCP đang hoạt động, tuy nhiên, các địa chỉ và số cổng được biểu thị dưới dạng số chứ không thể xác định tên cụ thể. |
| -o | Hiển thị các kết nối TCP đang hoạt động và bao gồm cả ID tiến trình (Process ID - PID) cho mỗi kết nối. Bạn có thể tìm ứng dụng bằng cách tra cứu PID trên tab Processes trong Windows Task Manager. Tham số này có thể được kết hợp với -a, -n và -p. |
| -p | Hiển thị kết nối cho giao thức được chỉ định bởi Protocol. Trong trường hợp này, Protocol có thể là tcp, udp, tcpv6 hoặc udpv6. Nếu tham số này được sử dụng với -s để hiển thị số liệu thống kê theo giao thức, Protocolcó thể là tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6 hoặc ipv6. |
| -s | Hiển thị số liệu thống kê theo giao thức. Theo mặc định, các số liệu thống kê được hiển thị cho các giao thức TCP, UDP, ICMP và IP. Nếu giao thức IPv6 được cài đặt, các thống kê sẽ được hiển thị cho giao thức TCP thông qua IPv6, UDP qua IPv6, ICMPv6 và IPv6. Tham số -p có thể được sử dụng để chỉ định một tập các giao thức. |
| -r | Hiển thị nội dung của bảng định tuyến IP (IP routing table). Thông tin này tương đương với lệnh in route.  Sau mỗi giây, thông tin được chọn sẽ được hiển thị lại. Nhấn CTRL + C để dừng quá trình hiển thị lại. Nếu tham số này bị bỏ qua, netstat chỉ in thông tin đã chọn một lần. |
| /? | Hiển thị trợ giúp tại command prompt. |

Bảng 1: Danh sách tham chiếu của lệnh netstat

* Chú giải
* Các tham số được sử dụng với lệnh này phải được bắt đầu bằng dấu gạch nối (-) thay vì dấu gạch chéo (/).
* Lệnh netstat cung cấp số liệu thống kê cho những đối tượng sau đây:
* Tên của giao thức (TCP hoặc UDP).
* Địa chỉ cục bộ. Địa chỉ IP của máy tính cục bộ và số cổng đang được sử dụng. Tên của máy tính cục bộ tương ứng với địa chỉ IP và tên của cổng được hiển thị trừ khi tham số -n được chỉ định. Nếu cổng chưa được thiết lập, số cổng được hiển thị dưới dạng dấu hoa thị (\*).
* Địa chỉ từ xa. Địa chỉ IP và số cổng của máy tính từ xa được kết nối. Các tên máy tính từ xa tương ứng với địa chỉ IP và cổng được hiển thị trừ khi tham số -n được chỉ định. Nếu cổng chưa được thiết lập, số cổng được hiển thị dưới dạng dấu hoa thị (\*).
* Trạng thái. Cho biết trạng thái của các kết nối TCP. Các trạng thái có thể như sau: CLOSE\_WAIT CLOSED ESTABLISHED FIN\_WAIT\_1 FIN\_WAIT\_2 LAST\_ACK listEN SYN\_RECEIVED SYN\_SEND timeD\_WAIT để biết thêm thông tin về trạng thái của kết nối TCP, tham khảo Rfc 793.
* Lệnh này chỉ có thể sử dụng nếu giao thức Internet Protocol (TCP/IP) được cài đặt như một thành phần trong thuộc tính của bộ điều hợp mạng trong hệ thông kết nối Mạng (Network Connections).

CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

1. Yêu cầu

Xây dựng chương trình my\_netstat với các tùy chọn sau:

|  |  |
| --- | --- |
| Tuỳ chọn | Mô tả |
| my\_netstat | Hiển thị tất cả các kết nối TCP đang hoạt động |
| my\_netstat -p tcp -a |
| my\_netstat -a | Hiển thị tất cả các kết nối TCP đang hoạt động cũng như các cổng TCP và UDP trên đó máy tính đang lắng nghe |
| my\_netstat -p udp -a | Hiển thị tất cả các cổng UDP mà trên đó máy tính đang lắng nghe |
| my\_netstat -s | Hiển thị số liệu thống kê theo giao thức. Theo mặc định, các số liệu thống kê được hiển thị cho các giao thức TCP, UDP, ICMP và IP. Nếu giao thức IPv6 được cài đặt, các thống kê sẽ được hiển thị cho giao thức TCP thông qua IPv6, UDP qua IPv6, ICMPv6 và IPv6. |

Bảng 2: Danh sách tùy chọn của chương trình my\_netstat

1. Phương pháp triển khai

* Hiển thị tất cả các kết nối TCP đang hoạt động
* Sử dụng hàm GetTcpTable() để lấy thông tin bảng kết nối TCP
* Cú pháp:



* Hiển thị các cổng UDP trên đó máy tính đang lắng nghe
* Sử dụng hàm GetUdpTable() để lấy thông tin tất cả các cổng UDP mà trên đó máy tính đang lắng nghe.
* Cú pháp



* Hiển thị số liệu thống kê theo giao thức
* Sử dụng các hàm GetIpStatisticsEx(), GetIcmpStatisticsEx(), GetTcpStatisticsEx(), GetUdpStatisticsEx() để lấy số liệu thống kê theo gian thức.
* Cú pháp
* Hàm GetIpStatisticsEx()



* Hàm GetIcmpStatisticsEx()



* Hàm GetTcpStatisticsEx()



* Hàm GetUdpStatisticsEx()



CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

1. Cài đặt chương trình

* Hàm lấy thông tin các kết nối TCP đang hoạt động



* Hàm lấy thông tin các cổng UDP trên đó máy tính đang lắng nghe



* Hàm lấy thông tin thống kê theo giao thức
* IP





* ICMP





* TCP





* UDP

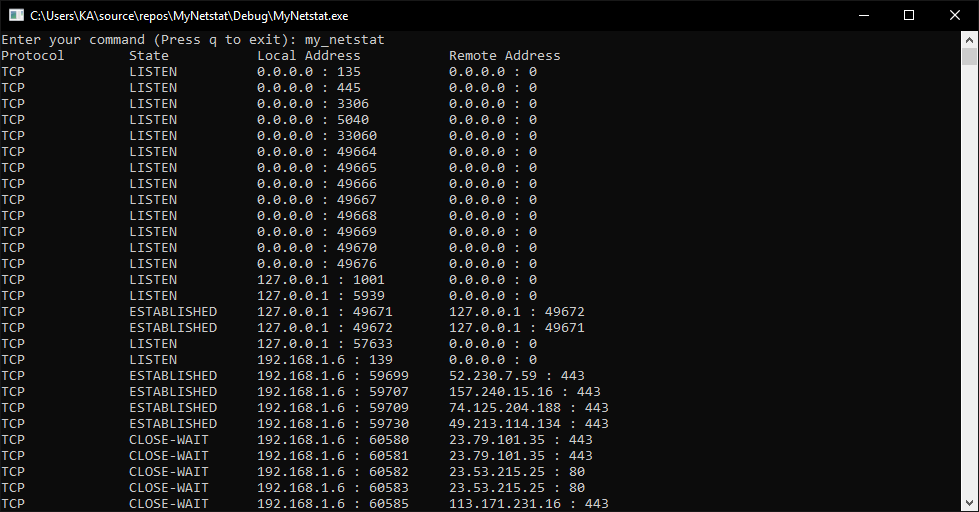




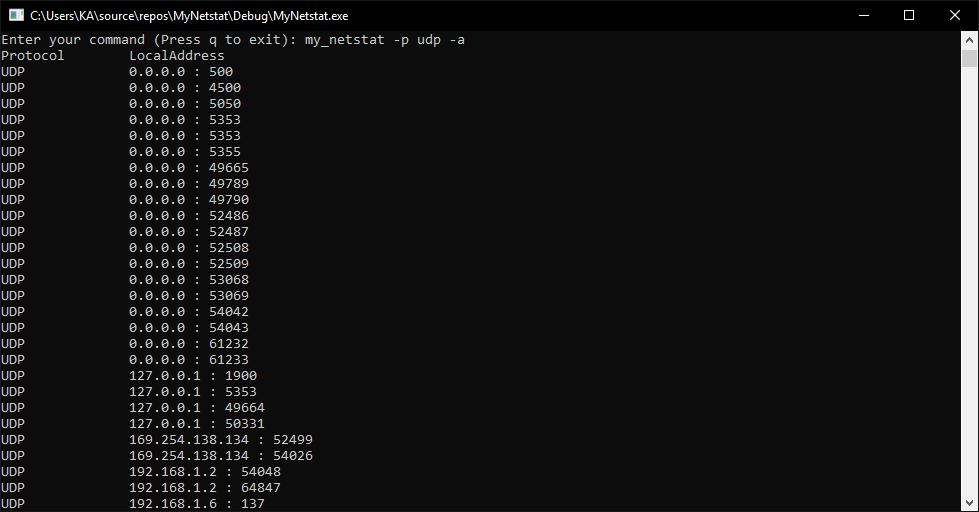
* Chương trình chính



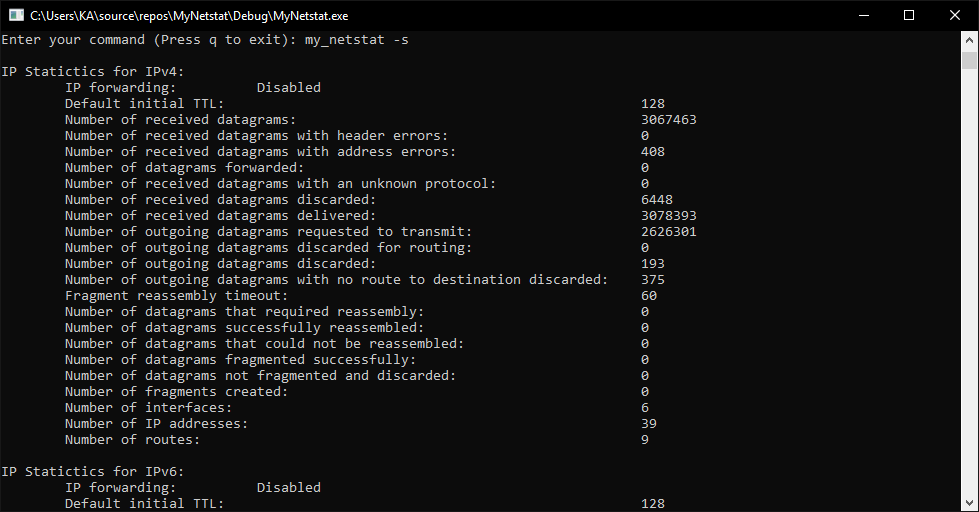
1. Kết quả chạy chương trình



Hình 9: Kết quả chạy chương trình với tùy chọn my\_netstat



Hình 10: Kết quả chạy chương trình với tùy chọn my\_netstat –p udp -a



Hình 11: Kết quả chạy chương trình với tùy chọn my\_netstat -s

1. Đánh giá kết quả

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN