

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN



**BÁO CÁO ĐỒ ÁN
CƠ SỞ NGÀNH MẠNG**

ĐỀ TÀI

- 1. Mô phỏng các giải thuật lập lịch trên hệ thống thời gian thực**
- 2. Tìm hiểu IP Helper và xây dựng chương trình my_netstat**



GVHD: ThS.Nguyễn Thế Xuân Ly
SVTH: Nguyễn Văn Mẫn
MSSV: 102150113

Đà Nẵng, 2018

[illegible]

Mục lục

DANH MỤC HÌNH VẼ.....	4
DANH SÁCH BẢNG BIỂU.....	4
MỞ ĐẦU.....	5
PHẦN I: NGUYÊN LÝ HỆ ĐIỀU HÀNH.....	6
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT.....	6
1. Hệ điều hành thời gian thực.....	6
2. Lập lịch thời gian thực.....	7
3. Một số thuật toán lập lịch thời gian thực.....	10
• Thuật toán Deadline Monotonic.....	10
• Thuật toán Rate Monotonic.....	11
• Thuật toán Earlier Deadline First.....	12
CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG.....	13
1. Yêu cầu.....	13
2. Phân tích và thiết kế hệ thống.....	13
• Sơ đồ hoạt động của các thuật toán:.....	13
• Môi trường triển khai.....	14
CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ.....	14
1. Cài đặt chương trình.....	14
• Lớp Task.....	14
• Nhập danh sách tác vụ.....	15
• Tính CPU Utilization.....	15
• Tính Hyper Period.....	16
• Khởi tạo danh sách các tác vụ sẵn sàng theo độ ưu tiên.....	16
▪ Deadline Monotonic.....	16
▪ Rate Monotonic.....	16
▪ Earlier Deadline First.....	17
• Xử lý lập lịch.....	17
2. Kết quả thực hiện.....	18
3. Đánh giá kết quả.....	19
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....	19

PHẦN II: LẬP TRÌNH MẠNG	20
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT	20
1. IP Helper	20
• Tổng quát về IP Helper	20
• Mục đích của IP Helper.....	21
• Nơi áp dụng IP Helper	21
• Đối tượng phát triển	21
• Các yêu cầu về môi trường run-time.....	21
2. Windows Sockets 2	21
• Mục đích	21
• Đối tượng phát triển	22
• Các yêu cầu về môi trường run-time.....	22
3. Sử dụng IP Helper.....	22
4. Lệnh netstat	24
• Cú pháp.....	24
• Tham chiếu	24
• Chú giải	25
CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG	26
1. Yêu cầu	26
2. Phương pháp triển khai.....	26
CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ	30
1. Cài đặt chương trình	30
• Hàm lấy thông tin các kết nối TCP đang hoạt động	30
• Hàm lấy thông tin các cổng UDP trên đó máy tính đang lắng nghe.....	32
• Hàm lấy thông tin thống kê theo giao thức	33
• Chương trình chính	37
2. Kết quả chạy chương trình	38
3. Đánh giá kết quả	39
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	39
TÀI LIỆU THAM KHẢO	40

DANH MỤC HÌNH VẼ

Hình 1: Các trạng thái của một tác vụ	8
Hình 2: Ví dụ cho thuật toán Deadline Monotonic	11
Hình 3: Ví dụ về thuật toán Rate Monotonic	12
Hình 4: Ví dụ về thuật toán Earlier Deadline First.....	13
Hình 5: Sơ đồ hoạt động của các thuật toán.....	14
Hình 6: Kết quả chương trình Deadline Monotonic	18
Hình 7: Kết quả chương trình Rate Monotonic.....	18
Hình 8: Kết quả chương trình Earlier Deadline First	19
Hình 9: Kết quả chạy chương trình với tùy chọn my_netstat	38
Hình 10: Kết quả chạy chương trình với tùy chọn my_netstat -p udp -a	38
Hình 11: Kết quả chạy chương trình với tùy chọn my_netstat -s.....	39

DANH SÁCH BẢNG BIỂU

Bảng 1: Danh sách tham chiếu của lệnh netstat	25
Bảng 2: Danh sách tùy chọn của chương trình my_netstat.....	26

MỞ ĐẦU

Đồ án cơ sở ngành mạng là một trong những đồ án quan trọng trong chương trình học của sinh viên ngành Công nghệ thông tin – trường Đại học Bách Khoa – Đại học Đà Nẵng. Quá trình tìm hiểu và thực hiện đồ án này sẽ giúp sinh viên chúng em hiểu rõ hơn về những kiến thức đã học trong các môn học Nguyên lý hệ điều hành và Mạng máy tính cũng như học thêm được nhiều kiến thức mới và cách để ứng dụng các kiến thức đã học vào các chương trình hữu ích trong thực tế.

Trong bài báo cáo này, em trình bày kết quả thực hiện các đề tài em đã được nhận từ đầu học kì. Về phần Nguyên lý hệ điều hành là đề tài: “Mô phỏng các giải thuật lập lịch trên hệ thống thời gian thực”, và phần Lập trình mạng là đề tài: “Tìm hiểu về IP Helper và xây dựng chương trình my_netstat”.

Sau thời gian gần một học kì, em đã tìm hiểu và hoàn thành được các đề tài đã nhận đúng với thời hạn. Em xin gửi lời cảm ơn trân trọng đến các thầy, cô trong bộ môn đã hỗ trợ và giúp đỡ em trong suốt quá trình làm đồ án.

Đặc biệt, em xin chân thành cảm ơn Thầy Nguyễn Thế Xuân Ly, người trực tiếp hướng dẫn, cung cấp tài liệu cũng như đưa ra những nhận xét một cách thẳng thắn để giúp em hoàn thành đồ án đúng với tiến độ.

Sinh viên thực hiện

PHẦN I: NGUYÊN LÝ HỆ ĐIỀU HÀNH

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

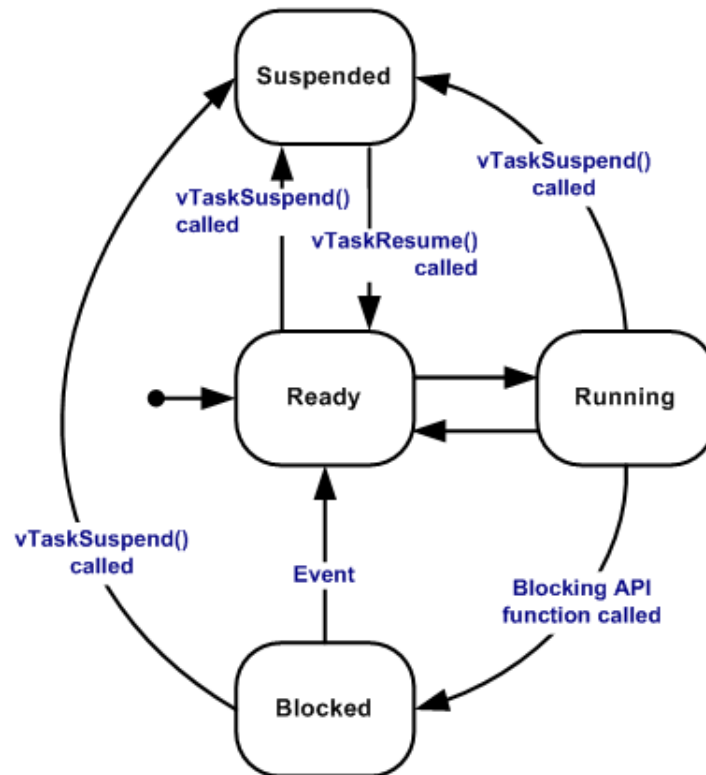
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

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 lý 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.

2. 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 hợp xấu nhất của tác vụ (WCET): C
- Sự dùng CPU (CPU Utilization):

- CPU Utilization của một tác vụ: $\frac{C}{P}$

- CPU Utilization của tập hợp các tác vụ: $U = \sum_{i=1}^n \frac{C_i}{P_i}$,

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

- CPU Utilization là thước đo sự bận rộn của bộ xử lý có thể lặp lại trong chu kỳ ngắn nhất:
 - $U > 1$ (Quá tải – Overload): một số tác vụ sẽ không đáp ứng được thời hạn.

- $U \leq 1$: 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.

3. 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 sàng 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

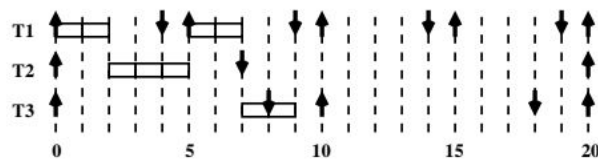
$$\sum_{i=1}^n \frac{c_i}{d_i} \leq n * \left(2^{\frac{1}{n}} - 1\right),$$

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

- Tasks characteristics

	WCET	Critical delay	Period
T_1	2	4	5
T_2	3	7	20
T_3	2	8	10

- Scheduling sequence in case of synchronous wake up



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 sẻ 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:

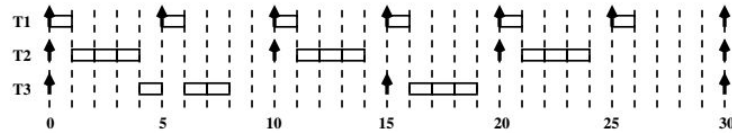
$$U = \sum_{i=1}^n \frac{C_i}{P_i} \leq n * \left(2^{\frac{1}{n}} - 1\right),$$

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

- Tasks characteristics

	WCET	Critical delay	Period
T_1	1	5	5
T_2	3	10	10
T_3	3	15	15

- Scheduling sequence in case of synchronous wake up



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 ràng buộc sử dụng cho thuật toán Earlier Deadline First:

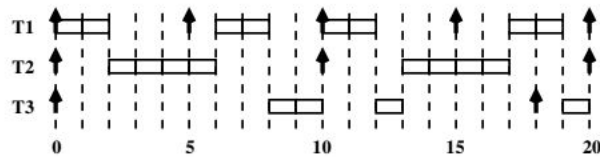
$$U = \sum_{i=1}^n \frac{C_i}{P_i} \leq 1,$$

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

- Tasks characteristics

	WCET	Critical delay	Period
T_1	2	5	5
T_2	4	10	10
T_3	3	18	18

- Scheduling sequence in case of synchronous wake up



Hình 4: Ví dụ về thuật toán Earlier Deadline First

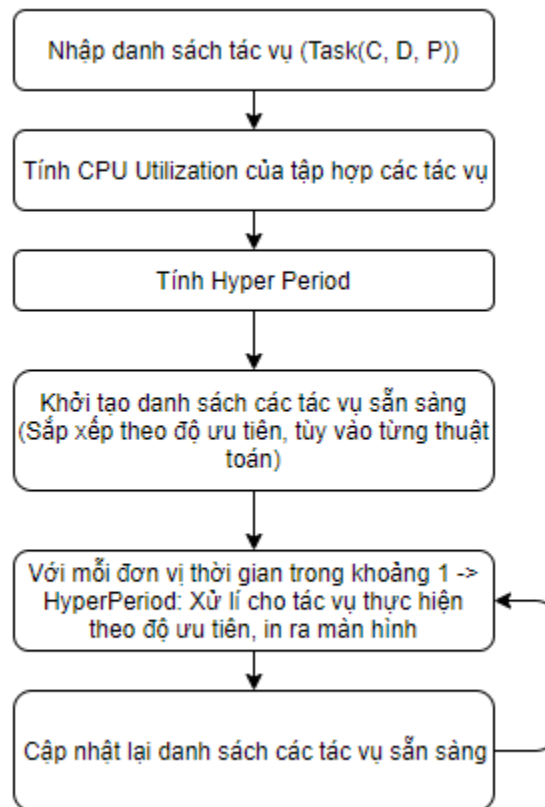
CHƯƠNG 2: PHÂN TÍCH THIẾT KẾ HỆ THỐNG

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

2. 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: Visual Studio 2017 Community.

CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

1. Cài đặt chương trình

- Lớp Task

```

public class Task
{
    public string Name { get; set; }
    public int Capacity { get; set; }
    public int Deadline { get; set; }
    public int Period { get; set; }
}
  
```

- Nhập danh sách tác vụ
 - Khởi tạo danh sách tác vụ

```
private static List<Task> Tasks = new List<Task>();
```

- Nhập danh sách tác vụ

```
private static void InputTasks()
{
    Console.WriteLine("Enter the number of tasks: ");
    var numberOfTasks = Convert.ToInt32(Console.ReadLine());

    for (var i = 0; i < numberOfTasks; i++)
    {
        Console.WriteLine($"Enter the information of task {i + 1}:");
        Console.WriteLine("Capacity: ");
        var capacity = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Deadline: ");
        var deadline = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Period: ");
        var period = Convert.ToInt32(Console.ReadLine());

        Tasks.Add(new Task
        {
            Name = $"Task {i + 1}",
            Capacity = capacity,
            Deadline = deadline,
            Period = period
        });
    }
}
```

- Tính CPU Utilization

```
private static float CalculateUtilization()
{
    var u = 0f;
    foreach (var task in Tasks)
    {
        u += task.Capacity * 1f / task.Period;
    }

    return u;
}
```


- Tính Hyper Period

```
private static int CalculateHyperPeriod()
{
    var hyper = 1;
    foreach (var task in Tasks)
    {
        hyper = Lcm(hyper, task.Period);
    }
    return hyper;
}
```

- Khởi tạo danh sách các tác vụ sẵn sàng theo độ ưu tiên

- Deadline Monotonic

```
private static void InitReadyTaskForDeadLineMonotonic()
{
    readyTasks = Tasks
        .OrderBy(x => x.Deadline)
        .Where(x => x.Deadline != 0)
        .Select(x => new ReadyTask
        {
            Task = x,
            CurrentCapacity = x.Capacity
        })
        .ToList();
}
```

- Rate Monotonic

```
private static void InitReadyTaskForRateMonotonic()
{
    readyTasks = Tasks
        .OrderBy(x => x.Period)
        .Select(x => new ReadyTask
        {
            Task = x,
            CurrentCapacity = x.Capacity
        })
        .ToList();
}
```

- Earlier Deadline First

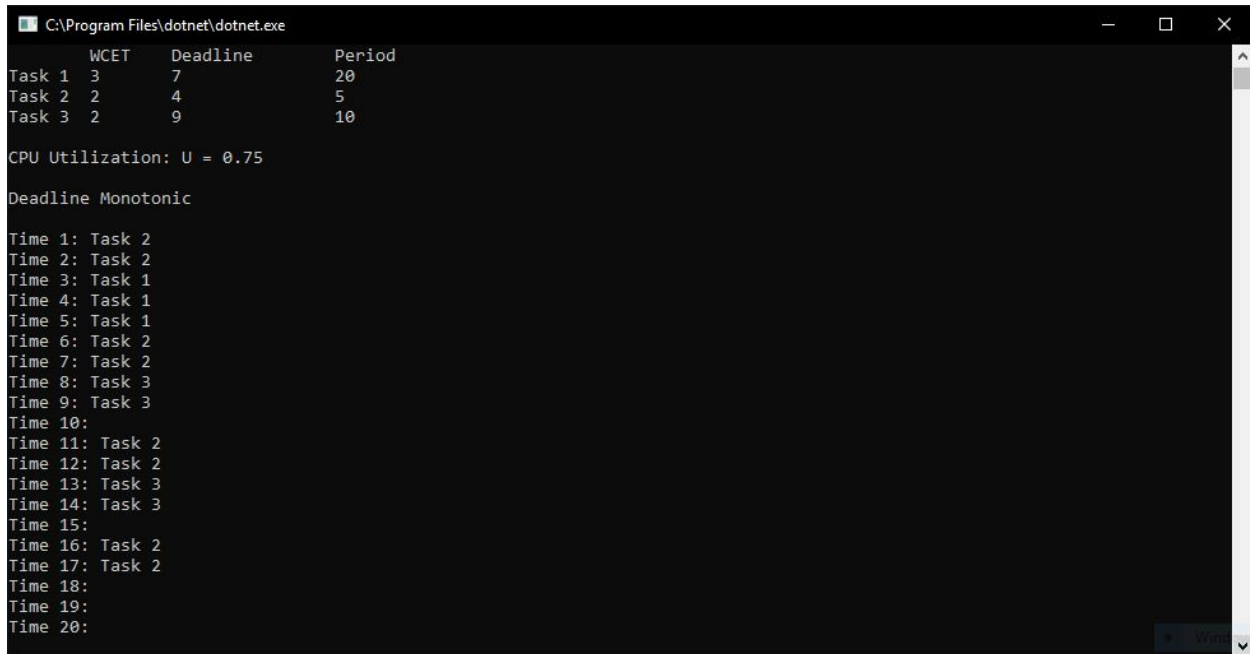
```
private static void InitReadyTaskForEarlierDeadlineFirst()
{
    readyTasks = Tasks.OrderBy(x => x.Deadline)
        .Where(x => x.Deadline != 0)
        .Select(x => new ReadyTask
        {
            Task = x,
            CurrentCapacity = x.Capacity,
            CurrentDeadline = x.Deadline
        })
        .ToList();
}
```

- Xử lý lập lịch

```
private static void Scheduling(Action initAction, bool isEdf)
{
    initAction.Invoke();
    for (var i = 1; i <= CalculateHyperPeriod(); i++)
    {
        Console.WriteLine("Time " + i + ": ");
        var index = 0;
        var isProcessing = false;
        while (index < readyTasks.Count)
        {
            if (readyTasks[index].CurrentCapacity > 0)
            {
                Console.WriteLine(readyTasks[index].Task.Name);
                readyTasks[index].CurrentCapacity--;
                isProcessing = true;
                if (isEdf) UpdateCurrentDeadline(i);
                break;
            }
            index++;
        }
        if (!isProcessing) Console.WriteLine();

        foreach (var t in readyTasks)
        {
            if (i % t.Task.Period == 0)
                t.CurrentCapacity = t.Task.Capacity;
        }
    }
}
```

2. Kết quả thực hiện



```
C:\Program Files\dotnet\dotnet.exe
WCET  Deadline  Period
Task 1  3        7        20
Task 2  2        4        5
Task 3  2        9        10

CPU Utilization: U = 0.75

Deadline Monotonic

Time 1: Task 2
Time 2: Task 2
Time 3: Task 1
Time 4: Task 1
Time 5: Task 1
Time 6: Task 2
Time 7: Task 2
Time 8: Task 3
Time 9: Task 3
Time 10:
Time 11: Task 2
Time 12: Task 2
Time 13: Task 3
Time 14: Task 3
Time 15:
Time 16: Task 2
Time 17: Task 2
Time 18:
Time 19:
Time 20:
```

Hình 6: Kết quả chương trình Deadline Monotonic



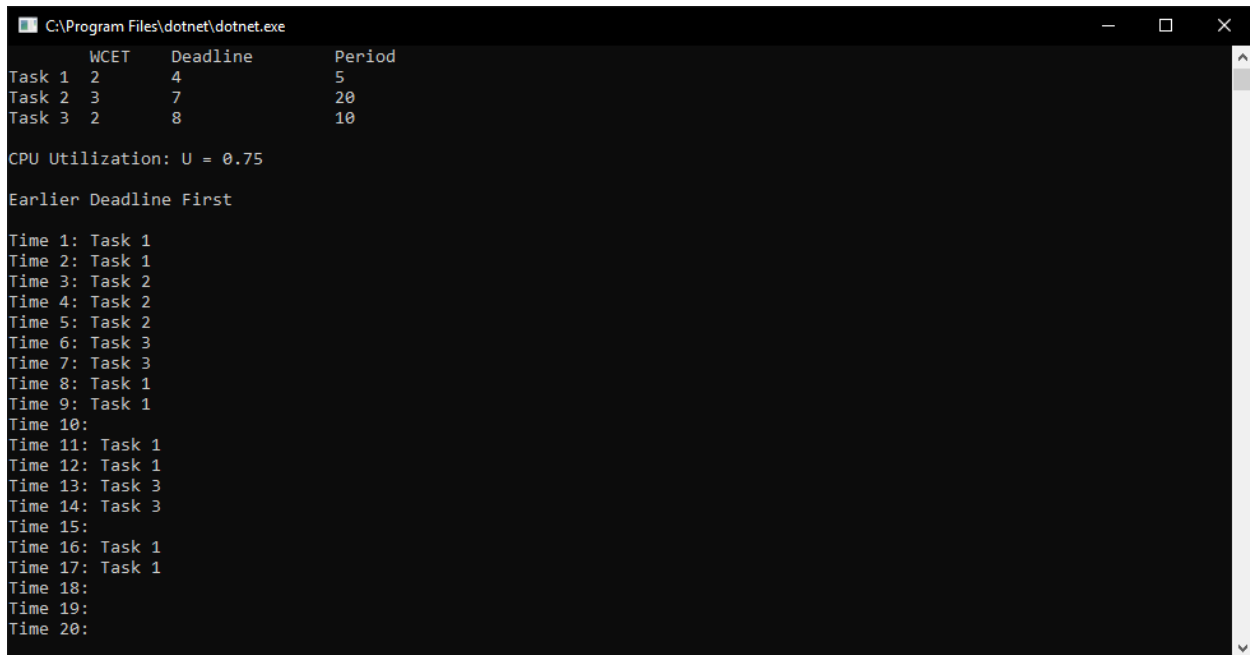
```
C:\Program Files\dotnet\dotnet.exe
WCET  Period
Task 1  2        5
Task 2  3       10
Task 3  6       20

CPU Utilization: U = 1

Rate Monotonic

Time 1: Task 1
Time 2: Task 1
Time 3: Task 2
Time 4: Task 2
Time 5: Task 2
Time 6: Task 1
Time 7: Task 1
Time 8: Task 3
Time 9: Task 3
Time 10: Task 3
Time 11: Task 1
Time 12: Task 1
Time 13: Task 2
Time 14: Task 2
Time 15: Task 2
Time 16: Task 1
Time 17: Task 1
Time 18: Task 3
Time 19: Task 3
Time 20: Task 3
```

Hình 7: Kết quả chương trình Rate Monotonic



```
C:\Program Files\dotnet\dotnet.exe
WCET  Deadline  Period
Task 1  2         4         5
Task 2  3         7        20
Task 3  2         8        10

CPU Utilization: U = 0.75

Earlier Deadline First

Time 1: Task 1
Time 2: Task 1
Time 3: Task 2
Time 4: Task 2
Time 5: Task 2
Time 6: Task 3
Time 7: Task 3
Time 8: Task 1
Time 9: Task 1
Time 10:
Time 11: Task 1
Time 12: Task 1
Time 13: Task 3
Time 14: Task 3
Time 15:
Time 16: Task 1
Time 17: Task 1
Time 18:
Time 19:
Time 20:
```

Hình 8: Kết quả chương trình Earlier Deadline First

3. Đánh giá kết quả

Các kết quả chạy chương trình ở trên đã đáp ứng được yêu cầu của bài toán đặt ra ban đầu, đó là mô phỏng lại cách thức các tác vụ được lập lịch trong một hệ điều hành thời gian thực bằng các thuật toán khác nhau: Deadline Monotonic, Rate Monotonic và Earlier Deadline First.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Quá trình tìm hiểu để hoàn thành đề tài này đã giúp em hiểu thêm một số kiến thức về hệ điều hành thời gian thực, cách thức hoạt động của một số thuật toán lập lịch trên hệ điều hành thời gian thực cũng như biết được cách thức để mô phỏng lại các thuật toán bằng các kiến thức đã học. Các thuật toán tuy đã hoạt động theo đúng như các nguyên tắc đặt ra nhưng về mặt hình thức, các chương trình vẫn còn hạn chế, giao diện sử dụng không thực sự thân thiện và khó để sử dụng. Trong tương lai, em sẽ tiếp tục phát triển để xây dựng giao diện cho chương trình thân thiện hơn và dễ sử dụng hơn để có cái nhìn trực quan và dễ hiểu hơn về cách mà các tác vụ được chia thời gian để thực thi trong một hệ điều hành thời gian thực.

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 LÝ 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 lý 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 lý bộ điều hợp mạng
 - Quản lý giao diện

- Quản lý 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 lý 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 lý 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.

2. 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 năng 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 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.

3. 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 rằng Build 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.

```
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#endif

#include <windows.h>

#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>

int main() {
    return 0;
}
```

- Một số lưu ý:
 - Tập 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 tập tiêu đề khác với các cấu trúc (structures) và các bảng liệt kê (enumerations) được sử dụng bởi IP Helper.
 - Tập 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>`
 - Tập tiêu đề winsock2.h bên trong bao gồm các thành phần chính từ tập 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 lý 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 tập winsock.h được bao gồm bởi tập windows.h.

4. 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

```
netstat [-a] [-e] [-n] [-o] [-p <Protocol>] [-r] [-s] [<Interval>]
```

- 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 <i>Protocol</i> . Trong trường hợp này, <i>Protocol</i> 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, <i>Protocol</i> có 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 <code>ip route</code> .

	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 thống 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:

Tùy 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

2. 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:

```

DWORD GetTcpTable(PMIB_TCPTABLE pTcpTable, PDWORD pdwSize, BOOL bOrder);
typedef struct _MIB_TCPTABLE
{
    DWORD dwNumEntries; // Specifies how many entries are in the table
    MIB_TCPROW table[ANY_SIZE]; // Is a pointer to an array of MIB_TCPROW
                                // structures that contain TCP connection information
} MIB_TCPTABLE, *PMIB_TCPTABLE;

typedef struct _MIB_TCPROW
{
    DWORD dwState; // Specifies the state of the TCP connection
    DWORD dwLocalAddr; // Specifies a local IPv4 address for the
    connection
    DWORD dwLocalPort; // Specifies a local port for the connection
    DWORD dwRemoteAddr; // Specifies the remote IPv4 address for the
    connection
    DWORD dwRemotePort; // Specifies the remote port for the connection
} MIB_TCPROW, *PMIB_TCPROW;

```

- Hiện 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

```
DWORD GetUdpTable(  
    PMIB_UDPTABLE pUdpTable,  
    PDWORD pdwSize,  
    BOOL bOrder  
);  
typedef struct _MIB_UDPTABLE  
{  
    DWORD dwNumEntries; //Specifies how many entries are in the table  
    field  
        MIB_UDPROW table[ANY_SIZE]; //Is a pointer to an array of MIB_UDPROW  
                                     //structures that contain UDP listener  
    information  
} MIB_UDPTABLE, *PMIB_UDPTABLE;  
typedef struct _MIB_UDPROW  
{  
    DWORD dwLocalAddr; //Specifies the local IPv4 address  
    DWORD dwLocalPort; //Specifies the local port  
} MIB_UDPROW, *PMIB_UDPROW;
```

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

```
ULONG GetIpStatisticsEx(
    PMIB_IPSTATS Statistics,
    ULONG          Family //The protocol family for which to retrieve
    statistics
    //AF_INET for Internet Protocol version 4 (IPv4)
    //AF_INET6 for Internet Protocol version 6 (IPv6)
);
typedef struct _MIB_IPSTATS_LH {
    union {
        DWORD          dwForwarding;
        MIB_IPSTATS_FORWARDING Forwarding;
    };
    DWORD dwDefaultTTL; //The default initial time-to-live (TTL) for
    datagrams originating on a particular computer
    DWORD dwInReceives; //The number of datagrams received
    DWORD dwInHdrErrors; //The number of datagrams received that have
    header errors
    DWORD dwInAddrErrors; //The number of datagrams received that have
    address errors
    DWORD dwForwDatagrams; //The number of datagrams forwarded
    DWORD dwInUnknownProtos; //The number of datagrams received that have
    an unknown protocol
    DWORD dwInDiscards; //The number of received datagrams discarded
    DWORD dwInDelivers; //The number of received datagrams delivered
    DWORD dwOutRequests; //The number of outgoing datagrams that IP is
    requested to transmit. This number does not include forwarded datagrams
    DWORD dwRoutingDiscards; //The number of outgoing datagrams discarded
    DWORD dwOutDiscards; //The number of transmitted datagrams discarded
    DWORD dwOutNoRoutes; //The number of datagrams for which this
    computer did not have a route to the destination IP address. These
    datagrams were discarded.
    DWORD dwReasmTimeout; //The amount of time allowed for all pieces of
    a fragmented datagram to arrive. If all pieces do not arrive within this
    time, the datagram is discarded
    DWORD dwReasmReqds; //The number of datagrams that require re-
    assembly
    DWORD dwReasmOks; //The number of datagrams that were successfully
    reassembled
    DWORD dwReasmFails; //The number of datagrams that cannot be
    reassembled
    DWORD dwFragOks; //The number of datagrams that were fragmented
    successfully
    DWORD dwFragFails; //The number of datagrams that have not been
    fragmented because the IP header specifies no fragmentation. These
    datagrams are discarded
    DWORD dwFragCreates; //The number of fragments created
    DWORD dwNumIf; //The number of interfaces
    DWORD dwNumAddr; //The number of IP addresses associated with this
    computer
    DWORD dwNumRoutes; //The number of routes in the IP routing table
} MIB_IPSTATS_LH, *PMIB_IPSTATS_LH;
```

▪ Hàm GetIcmpStatisticsEx()

```

ULONG GetIcmpStatisticsEx(
    PMIB_ICMP_EX Statistics,
    ULONG Family
);
typedef struct _MIB_ICMP_EX_XPSP1 {
    MIBICMPSTATS_EX icmpInStats;
    MIBICMPSTATS_EX icmpOutStats;
} MIB_ICMP_EX_XPSP1, *PMIB_ICMP_EX_XPSP1;
typedef struct _MIBICMPSTATS_EX_XPSP1 {
    DWORD dwMsgs; // Specifies the number of messages received or sent.
    DWORD dwErrors; // The number of errors received or sent.
    DWORD rgdwTypeCount[256]; // The type count
} MIBICMPSTATS_EX_XPSP1, *PMIBICMPSTATS_EX_XPSP1;

```

▪ Hàm GetTcpStatisticsEx()

```

ULONG GetTcpStatisticsEx(
    PMIB_TCPSTATS Statistics,
    ULONG Family
);
typedef struct _MIB_TCPSTATS_LH {
    union {
        DWORD dwRtoAlgorithm;
        TCP_RTO_ALGORITHM RtoAlgorithm;
    };
    DWORD dwRtoMin; // The minimum RTO value in milliseconds
    DWORD dwRtoMax; // The maximum RTO value in milliseconds
    DWORD dwMaxConn; // The maximum number of connections. If this member
    is -1, the maximum number of connections is variable.
    DWORD dwActiveOpens; // The number of active opens. In an active open,
    the client is initiating a connection with the server.
    DWORD dwPassiveOpens; // The number of passive opens. In a passive
    open, the server is listening for a connection request from a client.
    DWORD dwAttemptFails; // The number of failed connection attempts.
    DWORD dwEstabResets; // The number of established connections that
    were reset.
    DWORD dwCurrEstab; // The number of currently established connections.
    DWORD dwInSegs; // The number of segments received.
    DWORD dwOutSegs; // The number of segments transmitted. This number
    does not include retransmitted segments.
    DWORD dwRetransSegs; // The number of segments retransmitted.
    DWORD dwInErrs; // The number of errors received.
    DWORD dwOutRsts; // The number of segments transmitted with the reset
    flag set.
    DWORD dwNumConns; // The number of connections that are currently
    present in the system. This total number includes connections in all states
    except listening connections.
} MIB_TCPSTATS_LH, *PMIB_TCPSTATS_LH;

```

- Hàm GetUdpStatisticsEx()

```
ULONG GetUdpStatisticsEx(  
    PMIB_UDPSTATS Statistics,  
    ULONG Family  
);  
typedef struct _MIB_UDPSTATS {  
    DWORD dwInDatagrams; //The number of datagrams received.  
    DWORD dwNoPorts; //The number of datagrams received that were  
discarded because the port specified was invalid.  
    DWORD dwInErrors; //The number of erroneous datagrams received. This  
number does not include the value contained by the dwNoPorts member.  
    DWORD dwOutDatagrams; //The number of datagrams transmitted.  
    DWORD dwNumAddrs; //The number of entries in the UDP listener table.  
} MIB_UDPSTATS, *PMIB_UDPSTATS;
```

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

```

void retrievingTCPConnectionTable()
{
    PMIB_TCPTABLE pTcpTable;
    DWORD dwSize = 0, dwReturnValue = 0;
    char szLocalAddr[128], szRemoteAddr[128]; struct in_addr IpAddr;
    pTcpTable = (MIB_TCPTABLE *)malloc(sizeof(MIB_TCPTABLE));
    if (pTcpTable == NULL) {
        cout << ("Error allocating memory\n"); return; }
    dwSize = sizeof(MIB_TCPTABLE);
    if ((dwReturnValue = GetTcpTable(pTcpTable, &dwSize, TRUE)) ==
    ERROR_INSUFFICIENT_BUFFER) {
        free(pTcpTable);
        pTcpTable = (MIB_TCPTABLE *)malloc(dwSize);
        if (pTcpTable == NULL) {
            printf("Error allocating memory\n"); return; } }
    if ((dwReturnValue = GetTcpTable(pTcpTable, &dwSize, TRUE)) ==
    NO_ERROR) {
        wprintf(L"Protocol\tState\t\tLocal Address\t\tRemote
        Address\n");
        for (int i = 0; i < (int)pTcpTable->dwNumEntries; i++) {
            IpAddr.S_un.S_addr = (u_long)pTcpTable->
            table[i].dwLocalAddr;
            strcpy_s(szLocalAddr, sizeof(szLocalAddr),
            inet_ntoa(IpAddr));
            IpAddr.S_un.S_addr = (u_long)pTcpTable->
            table[i].dwRemoteAddr;
            strcpy_s(szRemoteAddr, sizeof(szRemoteAddr),
            inet_ntoa(IpAddr));
            wprintf(L"TCP\t\t");
            switch (pTcpTable->table[i].dwState) {
                case MIB_TCP_STATE_CLOSED: wprintf(L"CLOSED\t\t");
                break;
                case MIB_TCP_STATE_LISTEN: wprintf(L"LISTEN\t\t");
                break;
                case MIB_TCP_STATE_SYN_SENT: wprintf(L"SYN-SENT\t");
                break;
                case MIB_TCP_STATE_SYN_RCVD: wprintf(L"SYN-
                RECEIVED\t"); break;
                case MIB_TCP_STATE_ESTAB: wprintf(L"ESTABLISHED\t");
                break;
                case MIB_TCP_STATE_FIN_WAIT1: wprintf(L"FIN-WAIT-1\t");
                break;
                case MIB_TCP_STATE_FIN_WAIT2: wprintf(L"FIN-WAIT-
                2\t\t"); break;
                default: wprintf(L"UNKNOWN dwState value"); break;
            }
            wprintf(L"%hs : %u\t", szLocalAddr,
            ntohs((u_short)pTcpTable->table[i].dwLocalPort));
            wprintf(L"%hs : %u\n", szRemoteAddr,
            ntohs((u_short)pTcpTable->table[i].dwRemotePort));
        }
    }
    else {
        printf("\tGetTcpTable failed with %d\n", dwReturnValue);
        free(pTcpTable); return; }
    if (pTcpTable != NULL) {
        free(pTcpTable); pTcpTable = NULL; }
}

```


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

```

void retrievingUDPListenerTable()
{
    PMIB_UDPTABLE pUdpTable;
    DWORD dwSize = 0;
    DWORD dwReturnValue = 0;
    char szLocalAddr[128];
    struct in_addr IpAddr;
    pUdpTable = (MIB_UDPTABLE *)malloc(sizeof(MIB_UDPTABLE));
    if (pUdpTable == NULL) {
        printf("Error allocating memory\n");
        return;
    }
    dwSize = sizeof(MIB_UDPTABLE);
    if ((dwReturnValue = GetUdpTable(pUdpTable, &dwSize, TRUE)) ==
    ERROR_INSUFFICIENT_BUFFER) {
        free(pUdpTable);
        pUdpTable = (MIB_UDPTABLE *)malloc(dwSize);
        if (pUdpTable == NULL) {
            printf("Error allocating memory\n");
            return;
        }
    }
    if ((dwReturnValue = GetUdpTable(pUdpTable, &dwSize, TRUE)) ==
    NO_ERROR)
    {
        wprintf(L"Protocol\tLocalAddress\n");
        for (int i = 0; i < (int)pUdpTable->dwNumEntries; i++) {
            wprintf(L"UDP\t");
            IpAddr.S_un.S_addr = (u_long)pUdpTable->
            table[i].dwLocalAddr;
            strcpy_s(szLocalAddr, sizeof(szLocalAddr),
            inet_ntoa(IpAddr));
            wprintf(L"\t%hs : %u\n", szLocalAddr,
            ntohs((u_short)pUdpTable->table[i].dwLocalPort));
        }
    }
}

```



```
void showIpStatistics()  
{  
    wprintf(L"\nIP Statistics for IPv4: \n");  
    retrievingIpStatistics(AF_INET);  
    wprintf(L"\nIP Statistics for IPv6: \n");  
    retrievingIpStatistics(AF_INET6);  
}
```

○ ICMP

```
void retrievingIcmpStatistics(ULONG family)  
{  
    DWORD dwReturnValue = 0;  
    PMIB_ICMP_EX icmpStatistics;  
    icmpStatistics = (MIB_ICMP_EX *)malloc(sizeof(MIB_ICMP_EX));  
    if (icmpStatistics == NULL) {  
        wprintf(L"Error allocating memory");  
        return;  
    }  
    dwReturnValue = GetIcmpStatisticsEx(icmpStatistics, family);  
    if (dwReturnValue != NO_ERROR) {  
        wprintf(L"GetIcmpStatistics call failed with %d\n",  
dwReturnValue);  
    }  
    else {  
        wprintf(L"\tNumber of incoming ICMP messages: %ld\n",  
icmpStatistics->icmpInStats.dwMsgs);  
        wprintf(L"\tNumber of incoming ICMP errors received: %ld\n",  
icmpStatistics->icmpInStats.dwErrors);  
        wprintf(L"\tNumber of outgoing ICMP messages: %ld\n",  
icmpStatistics->icmpOutStats.dwMsgs);  
        wprintf(L"\tNumber of outgoing ICMP errors received: %ld\n",  
icmpStatistics->icmpOutStats.dwErrors);  
    }  
    if (icmpStatistics) {  
        free(icmpStatistics);  
    }  
}
```

```
void showIcmpStatistics()  
{  
    wprintf(L"\nICMP Statistics for IPv4: \n");  
    retrievingIcmpStatistics(AF_INET);  
    wprintf(L"\nICMP Statistics for IPv6: \n");  
    retrievingIcmpStatistics(AF_INET6);  
}
```

○ TCP

```
void retrievingTcpStatistics(ULONG family)
{
    DWORD dwReturnValue = 0;
    PMIB_TCPSTATS tcpStatistics;
    tcpStatistics = (MIB_TCPSTATS *)malloc(sizeof(MIB_IPSTATS));
    if (tcpStatistics == NULL) {
        wprintf(L"Error allocating memory");
        return;
    }
    dwReturnValue = GetTcpStatisticsEx(tcpStatistics, family);
    if (dwReturnValue != NO_ERROR) {
        wprintf(L"GetTcpStatistics call failed with %d\n",
dwReturnValue);
    }
    else {
        wprintf(L"\tActive Opens: %ld\n", tcpStatistics->dwActiveOpens);
        wprintf(L"\tPassive Opens: %ld\n", tcpStatistics->dwPassiveOpens);
        wprintf(L"\tSegments Recv: %ld\n", tcpStatistics->dwInSegs);
        wprintf(L"\tSegments Xmit: %ld\n", tcpStatistics->dwOutSegs);
        wprintf(L"\tTotal # Conxs: %ld\n", tcpStatistics->dwNumConns);
    }
    if (tcpStatistics) {
        free(tcpStatistics);
    }
}

void showTcpStatistics()
{
    wprintf(L"\nTCP Statistics for IPv4: \n");
    retrievingTcpStatistics(AF_INET);
    wprintf(L"\nTCP Statistics for IPv6: \n");
    retrievingTcpStatistics(AF_INET6);
}
```

○ UDP

```
void retrievingUdpStatistics(ULONG family)
{
    DWORD dwReturnValue = 0;
    PMIB_UDPSTATS udpStatistics;
    udpStatistics = (MIB_UDPSTATS *)malloc(sizeof(MIB_UDPSTATS));
    if (udpStatistics == NULL) {
        wprintf(L"Error allocating memory");
        return;
    }
    dwReturnValue = GetUdpStatisticsEx(udpStatistics, family);
    if (dwReturnValue != NO_ERROR) {
        wprintf(L"GetUdpStatistics call failed with %d\n",
dwReturnValue);
    }
    else {
        printf("\tNumber of datagrams received: %ld\n", udpStatistics->dwInDatagrams);
        printf("\tNumber of datagrams discarded because the port
number was bad: %ld\n", udpStatistics->dwNoPorts);
        printf("\tNumber of erroneous datagrams received: %ld\n",
udpStatistics->dwInErrors);
        printf("\tNumber of datagrams transmitted: %ld\n",
udpStatistics->dwOutDatagrams);
    }
    if (udpStatistics) {
        free(udpStatistics);
    }
}

void showUdpStatistics()
{
    wprintf(L"\nUDP Statistics for IPv4: \n");
    retrievingUdpStatistics(AF_INET);
    wprintf(L"\nUDP Statistics for IPv6: \n");
    retrievingUdpStatistics(AF_INET6);
}
```

- Chương trình chính

```
int main()
{
    char commandLine[20];
    while(true) {
        printf("Enter your command (Press q to exit): ");
        scanf("%[^\n]*c", commandLine);

        if (strcmp("q", commandLine) == 0 || strcmp("Q", commandLine)
== 0) {
            break;
        }
        if (strcmp("my_netstat", commandLine) == 0 ||
strcmp("my_netstat -p tcp -a", commandLine) == 0) {
            retrievingTCPConnectionTable();
        }
        else if (strcmp("my_netstat -a", commandLine) == 0) {
            retrievingTCPConnectionTable();
            retrievingUDPListenerTable();
        }
        else if (strcmp("my_netstat -p udp -a", commandLine) == 0) {
            retrievingUDPListenerTable();
        }
        else if (strcmp("my_netstat -s", commandLine) == 0)
        {
            showIpStatistics();
            showIcmpStatistics();
            showTcpStatistics();
            showUdpStatistics();
        }
        else {
            printf("\nPlease enter the valid command line!\n");
        }
    }
    system("pause");
}
```

2. Kết quả chạy chương trình

```

C:\Users\KA\source\repos\MyNetstat\Debug\MyNetstat.exe
Enter your command (Press q to exit): my_netstat
Protocol    State      Local Address      Remote Address
TCP         LISTEN     0.0.0.0 : 135           0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 445           0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 3306        0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 5040        0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 33060       0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 49664       0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 49665       0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 49666       0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 49667       0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 49668       0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 49669       0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 49670       0.0.0.0 : 0
TCP         LISTEN     0.0.0.0 : 49676       0.0.0.0 : 0
TCP         LISTEN     127.0.0.1 : 1001        0.0.0.0 : 0
TCP         LISTEN     127.0.0.1 : 5939        0.0.0.0 : 0
TCP         ESTABLISHED 127.0.0.1 : 49671     127.0.0.1 : 49672
TCP         ESTABLISHED 127.0.0.1 : 49672     127.0.0.1 : 49671
TCP         LISTEN     127.0.0.1 : 57633     0.0.0.0 : 0
TCP         LISTEN     192.168.1.6 : 139      0.0.0.0 : 0
TCP         ESTABLISHED 192.168.1.6 : 59699    52.230.7.59 : 443
TCP         ESTABLISHED 192.168.1.6 : 59707    157.240.15.16 : 443
TCP         ESTABLISHED 192.168.1.6 : 59709    74.125.204.188 : 443
TCP         ESTABLISHED 192.168.1.6 : 59730    49.213.114.134 : 443
TCP         CLOSE-WAIT 192.168.1.6 : 60580    23.79.101.35 : 443
TCP         CLOSE-WAIT 192.168.1.6 : 60581    23.79.101.35 : 443
TCP         CLOSE-WAIT 192.168.1.6 : 60582    23.53.215.25 : 80
TCP         CLOSE-WAIT 192.168.1.6 : 60583    23.53.215.25 : 80
TCP         CLOSE-WAIT 192.168.1.6 : 60585    113.171.231.16 : 443

```

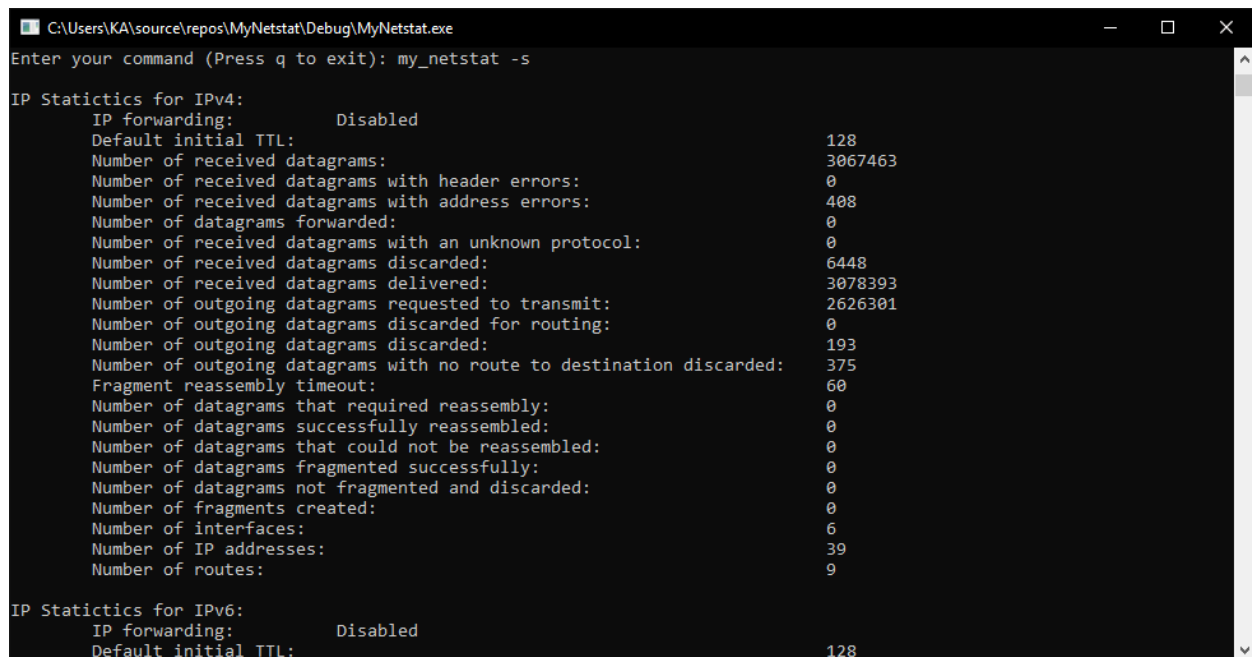
Hình 9: Kết quả chạy chương trình với tùy chọn my_netstat

```

C:\Users\KA\source\repos\MyNetstat\Debug\MyNetstat.exe
Enter your command (Press q to exit): my_netstat -p udp -a
Protocol    LocalAddress      Remote Address
UDP         0.0.0.0 : 500      0.0.0.0 : 0
UDP         0.0.0.0 : 4500     0.0.0.0 : 0
UDP         0.0.0.0 : 5050     0.0.0.0 : 0
UDP         0.0.0.0 : 5353     0.0.0.0 : 0
UDP         0.0.0.0 : 5353     0.0.0.0 : 0
UDP         0.0.0.0 : 5355     0.0.0.0 : 0
UDP         0.0.0.0 : 49665     0.0.0.0 : 0
UDP         0.0.0.0 : 49789     0.0.0.0 : 0
UDP         0.0.0.0 : 49790     0.0.0.0 : 0
UDP         0.0.0.0 : 52486     0.0.0.0 : 0
UDP         0.0.0.0 : 52487     0.0.0.0 : 0
UDP         0.0.0.0 : 52508     0.0.0.0 : 0
UDP         0.0.0.0 : 52509     0.0.0.0 : 0
UDP         0.0.0.0 : 53068     0.0.0.0 : 0
UDP         0.0.0.0 : 53069     0.0.0.0 : 0
UDP         0.0.0.0 : 54042     0.0.0.0 : 0
UDP         0.0.0.0 : 54043     0.0.0.0 : 0
UDP         0.0.0.0 : 61232     0.0.0.0 : 0
UDP         0.0.0.0 : 61233     0.0.0.0 : 0
UDP         127.0.0.1 : 1900     0.0.0.0 : 0
UDP         127.0.0.1 : 5353     0.0.0.0 : 0
UDP         127.0.0.1 : 49664     0.0.0.0 : 0
UDP         127.0.0.1 : 50331     0.0.0.0 : 0
UDP         169.254.138.134 : 52499 169.254.138.134 : 54026
UDP         192.168.1.2 : 54048 192.168.1.2 : 64847
UDP         192.168.1.6 : 137

```

Hình 10: Kết quả chạy chương trình với tùy chọn my_netstat -p udp -a



```
C:\Users\KA\source\repos\MyNetstat\Debug\MyNetstat.exe
Enter your command (Press q to exit): my_netstat -s

IP Statistics for IPv4:
  IP forwarding:           Disabled
  Default initial TTL:     128
  Number of received datagrams: 3067463
  Number of received datagrams with header errors: 0
  Number of received datagrams with address errors: 408
  Number of datagrams forwarded: 0
  Number of received datagrams with an unknown protocol: 0
  Number of received datagrams discarded: 6448
  Number of received datagrams delivered: 3078393
  Number of outgoing datagrams requested to transmit: 2626301
  Number of outgoing datagrams discarded for routing: 0
  Number of outgoing datagrams discarded: 193
  Number of outgoing datagrams with no route to destination discarded: 375
  Fragment reassembly timeout: 60
  Number of datagrams that required reassembly: 0
  Number of datagrams successfully reassembled: 0
  Number of datagrams that could not be reassembled: 0
  Number of datagrams fragmented successfully: 0
  Number of datagrams not fragmented and discarded: 0
  Number of fragments created: 0
  Number of interfaces: 6
  Number of IP addresses: 39
  Number of routes: 9

IP Statistics for IPv6:
  IP forwarding:           Disabled
  Default initial TTL:     128
```

Hình 11: Kết quả chạy chương trình với tùy chọn my_netstat -s

3. Đánh giá kết quả

Kết quả chạy chương trình đã mô phỏng lại một phần các chức năng của chương trình netstat và đã đáp ứng được yêu cầu đặt ra ban đầu của đề tài.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Đề tài này đã mang lại cho em thêm nhiều kiến thức mới về IP Helper cũng như cách vận dụng IP Helper để tạo ra các ứng dụng liên quan đến các giao thức mạng TCP/IP, UDP. Chương trình my_netsat đã được hoàn thành với một vài các tùy chọn được mô phỏng hoạt động giống với chương trình netstat. Tuy my_netstat chỉ là một chương trình mô phỏng nhưng trong tương lai, em sẽ tiếp tục phát triển thêm các tùy chọn khác để giống với chương trình netstat cũng là cơ hội để tìm hiểu nhiều hơn về các ứng dụng của IP Helper và học thêm về cách sử dụng IP Helper trong ứng dụng của mình.

TÀI LIỆU THAM KHẢO

- IP Helper Functions 13 Part 2 - <https://www.winsocketdotnetworkprogramming.com/winsock2programming/winsock2advancediphelperfunction13a.html>
- IP Helper – Microsoft Documentation - <https://docs.microsoft.com/en-us/windows/desktop/iphlp/ip-helper-start-page>
- Real-time Operating System - https://en.wikipedia.org/wiki/Real-time_operating_system
- <https://csperskins.org/teaching/2013-2014/adv-os/>
- https://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling
- https://en.wikipedia.org/wiki/Deadline-monotonic_scheduling
- https://en.wikipedia.org/wiki/Rate-monotonic_scheduling
- Embedded Systems: some typical problems - Jean-Luc Scharbarg Universit'e de Toulouse - IRIT/ENSEEIH/INPT Jean-Luc.Scharbarg@enseeiht.fr