Sinh viên lựa chọn 1 trong các đề tài trong danh sách, thực hiện rồi làm báo cáo và slide trình bày.

I.Nội dung báo cáo:

1. Giới thiệu đề tài

- Trình bày khái quát về đề tài/ Bài toán mà đề tài giải quyết

2. Nội dung chính của đề tài

- Miêu tả các giải thuật/ Lời giải đã lựa chọn cho bài toán

3. Miêu tả chương trình

- Các hàm chính của chương trình:
- + Miêu tả dữ liệu đầu ra/ đầu vào

  + Công việc mà hàm thực hiện

4. Miêu tả giao diện chương trình (Optional)

   + Điểm cộng cho chương trình co giao diện đồ họa

5. Phân công công việc của các thành viên trong nhóm

   +Chỉ rõ công việc đã thực hiện: Ví dụ thành viên A thu thập tài liệu, vẽ hình 1.1 trong báo cáo, xây dựng hàm ABC trong chương trình.

6. Tài liệu tham khảo (Optional)

   Nếu chương trình của sinh viên có tham khảo từ các nguồn khác thì phải chỉ rõ những phần tham khảo. Nếu không chỉ ra sẽ nhận 0 điểm.

II.Nội dung Slide được trích rút từ báo cáo.

Slides trình bày trong khoảng 15 phút. (Không copy paste y nguyên đoạn văn đưa vào slides).

(**Thành viên trong nhóm không nắm rõ về nội dung đề tài của nhóm: Cả nhóm bị trừ 1 điểm**)

ĐỀ TÀI BÀI TẬP LỚN HỆ ĐIỀU HÀNH kỳ 2024-2

1. Tìm hiểu về RPC (Remote Procedure Calls) và xây dựng chương trình minh họa
   - Triển khai một hệ thống RPC đơn giản bằng ngôn ngữ bạn chọn.
   - Thảo luận về ưu và nhược điểm của RPC so với các cơ chế IPC khác.
   - Đánh giá hiệu suất của hệ thống RPC của bạn.
2. Tìm hiểu về kỹ thuật truyền thông chia sẻ bộ nhớ (shared memory) trong Windows và xây dựng chương trình minh họa gửi các thông điệp có kích thước không xác định hoặc tập tin giữa 2 tiến trình
   -Cài đặt giao tiếp bộ nhớ chia sẻ trong Windows
   -Thảo luận về những thách thức và giải pháp để quản lý bộ nhớ chia sẻ.
3. Tìm hiểu về kỹ thuật truyền thông chia sẻ bộ nhớ (shared memory) trong Linux và xây dựng chương trình minh họa gửi các thông điệp có kích thước không xác định hoặc tập tin giữa 2 tiến trình
   -Triển khai giao tiếp bộ nhớ chia sẻ trong Linux.
   -Thảo luận về những thách thức và giải pháp để quản lý bộ nhớ chia sẻ.
4. Xây dựng chương trình minh họa giải thuật lập lịch hồi tiếp đa mức (bao gồm của Linux và Windows).
   -Triển khai bộ lập lịch MLFQ và đánh giá hiệu suất của nó.
   -So sánh MLFQ với các thuật toán lập lịch khác.

-Thảo luận về các yếu tố ảnh hưởng đến hiệu suất của MLFQ.

5. Tìm hiểu và cài đặt các giải thuật lập lịch trong các hệ thống đa xử lý
   -Triển khai và đánh giá các thuật toán lập lịch khác nhau cho hệ thống đa xử lý.
   -Thảo luận về những thách thức của việc lập lịch trong hệ thống đa xử lý.
   -So sánh hiệu suất của các thuật toán khác nhau.
6. Tìm hiểu và cài đặt các giải thuật lập lịch trong các hệ thống thời gian thực
7. Xây dựng chương trình minh họa giải pháp cho bài toán Reader -Writer
8. Xây dựng chương trình minh họa giải pháp cho bài toán Bathroom
9. Xây dựng chương trình minh họa giải pháp cho bài toán Người thợ cắt tóc
10. Xây dựng chương trình minh họa giải pháp cho bài toán Barrier
11. Xây dựng chương trình minh họa giải pháp cho bài toán Bữa tối của triết gia
12. Xây dựng chương trình minh họa đồng bộ hóa tiến trình bằng Monitor (trên Java hoặc ngôn ngữ lập trình khác có hỗ trợ)
13. Xây dựng chương trình minh họa bán vé online. Hệ thống (server) có 1 số lượng vé hữu hạn (N vé) . Có M tiến trình Client cùng truy cập cùng lúc (M >> N). Đồng bộ hóa hoạt động của các tiến trình sao cho hệ thống không bán vé khi số lượng vé âm.
    -Triển khai một hệ thống phân tán để bán vé trực tuyến.
    -Đảm bảo rằng hệ thống có khả năng mở rộng và chịu lỗi.
    -Đánh giá hiệu suất của hệ thống.
14. Xây dựng chương trinh quản lí tiến trình trên Linux, các bước tạo, hủy tiến trình và xây dựng chương trình mình họa
    -Triển khai một trình quản lý tiến trình đơn giản trong Linux.
    -Chứng minh sự hiểu biết của bạn về việc tạo, chấm dứt và lập lịch tiến trình.
    -Thảo luận về những thách thức của việc quản lý tiến trình.
15. Xây dựng chương trình minh họa cho các giải thuật quản lý bộ nhớ theo phương pháp phân trang, phân đoạn
    -Triển khai phân trang, phân đoạn hoặc bộ nhớ ảo.
    -Đánh giá hiệu suất của các kỹ thuật quản lý bộ nhớ khác nhau.
    -Thảo luận về sự đánh đổi giữa các kỹ thuật khác nhau.
16. Xây dựng chương trình minh họa cho các giải thuật thay thế trang, đầu vào là chuỗi truy cập trang của chương trình, đầu ra là số lỗi trang của các giải thuật tương ứng.
17. Tìm hiểu về hệ thống file ext3,4 trên Linux và xây dựng chương trình minh họa. Sử dụng các hàm API để đọc thông tin về bản ghi file, vị trí của file trên đĩa
18. Tìm hiểu về hệ thống file Apple File System trên MacOs và xây dựng chương trình minh họa. Sử dụng các hàm API để đọc thông tin về bản ghi file, vị trí của file trên đĩa
19. Cài đặt và tìm hiểu về quản lí bộ nhớ trên Hệ Điều Hành Minix
20. Xây dựng clip hoạt họa trình bày về chủ đề (bất kỳ) trong hệ điều hành. Sinh viên lựa chọn chủ đề rồi gủi thông tin để giảng viên xác nhận.
21. System Call Implementation:
    Details: Create a set of custom system calls that perform unique operations not covered by standard system calls. Examples could include interacting with specialized hardware or implementing advanced file operations.

Hints: Understand the anatomy of system calls, the role of the syscall interface, and how parameters are passed between user and kernel space.

22. Kernel Module Development:
    Details: Develop kernel modules that extend the functionality of the operating system dynamically. This could involve adding support for new devices, filesystems, or security features.
    Hints: Familiarize yourself with the Linux kernel module framework or the equivalent in the operating system of your choice. Understand how to load, unload, and interact with kernel modules.
23. Concurrency Control in the Kernel:
    Details: Implement synchronization mechanisms within the kernel to manage concurrent access to shared resources. This could include designing and implementing locks, semaphores, or other synchronization primitives.
    Hints: Study the principles of concurrent programming, understand race conditions and critical sections, and explore how these concepts are applied at the kernel level.
24. Real-time Operating System (RTOS):
    Details: Develop or enhance an operating system to support real-time requirements, ensuring timely and predictable responses to external events.
    Hints: Learn about real-time scheduling algorithms, priority inversion, and techniques for minimizing latency. Consider how to guarantee deadlines for critical tasks.
25. Process and Thread Management:
    Details: Extend the process and thread management capabilities in the kernel. This might involve implementing features like lightweight processes, thread pooling, or user-level threads.
    Hints: Dive into the internals of process and thread management, understand context switching, and explore ways to optimize the creation and destruction of processes and threads.
26. Interrupt Handling Optimization:
    Details: Improve the efficiency of interrupt handling routines in the kernel, optimizing response times and minimizing disruptions to the system.
    Hints: Study interrupt handling mechanisms in the kernel, explore ways to reduce interrupt latency, and consider optimizations such as interrupt coalescing.
27. Memory Allocator Design:
    Details: Create a custom memory allocator for the operating system, optimizing for speed, fragmentation, and scalability.
    Hints: Understand different memory allocation algorithms (e.g., buddy allocation, slab allocation), study existing allocators in operating systems, and experiment with designing your own.
28. Containerization in the Kernel:
    Details: Explore the integration of containerization technologies at the kernel level, allowing for lightweight and isolated application environments.

Hints: Study containerization principles, understand container runtimes, and explore how namespaces and cgroups can be managed at the kernel level.

29. Dynamic Linker/Loader Implementation:
    Details: Implement a dynamic linker/loader for the operating system, enabling dynamic linking of shared libraries at runtime.
    Hints: Explore the ELF (Executable and Linkable Format) specification, understand how dynamic linking works, and consider implementing lazy loading and symbol resolution.
30. Fault Tolerance Mechanisms:
    Details: Introduce fault tolerance mechanisms at the kernel level, such as checkpointing, process migration, or recovery strategies to enhance system reliability.
    Hints: Study fault tolerance concepts, explore techniques like process checkpointing and migration, and consider how to recover from failures gracefully.

AI-related topics that can be explored in the context of operating systems:

31. AI Task Scheduling:
    Implement an intelligent task scheduler that uses machine learning algorithms to predict the resource needs of different tasks and optimally schedules them on the system.
32. Resource Allocation with Machine Learning:
    Develop a system that uses AI techniques to dynamically allocate system resources (CPU, memory, etc.) based on the requirements and usage patterns of running applications.
33. Autonomous System Management:
    Create an autonomous system manager that uses AI to monitor the health of the system, predict potential issues, and take proactive measures to prevent or mitigate problems.
34. Self-Optimizing Systems:
    Design an operating system that can autonomously optimize its configurations and parameters using reinforcement learning or other AI methods to improve performance and energy efficiency.
35. Intelligent Memory Management:
    Utilize machine learning to predict and adaptively manage memory allocation based on application behavior and usage patterns, optimizing for both performance and resource efficiency.
36. AI-Based Security Enhancements:
    Implement AI-driven intrusion detection and prevention mechanisms within the operating system to identify and respond to security threats in real-time.
37. Predictive File System Caching:
    Develop a file system that employs machine learning models to predict which files are likely to be accessed in the near future and proactively caches them for improved I/O performance.
38. Smart Power Management:

Use AI algorithms to analyze usage patterns and predict periods of low activity, allowing the operating system to intelligently adjust power settings to conserve energy without sacrificing performance.

39. Personalized User Experience:
    Create an operating system interface that learns and adapts to user preferences over time using AI, providing a more personalized and efficient user experience.
40. Distributed AI Processing:
    Explore how operating systems can facilitate the deployment and management of distributed AI applications, ensuring efficient communication and resource utilization across a network of machines.