Case Study #1

Resource Allocation and Deadlock Handling in a Multi-User Operating System

Jiading Zhou

CSYE 6230 – Operating System

College of Engineering, Northeastern University

February 4, 2025

Case Study Scenario:

In a hypothetical multi-user operating system named "UniOS," designed for academic institutions, various students, faculty, and administrative staff heavily rely on it for their daily tasks. UniOS provides essential services such as file management, process scheduling, and user authentication.

Recently, the system has been facing challenges related to resource allocation and deadlock handling. Users have reported slow response times and occasional system crashes. The UniOS development team suspects that these issues are related to how resources are allocated and how deadlock situations are handled.

Case Study Questions:

Resource Allocation Policies

UniOS uses a few resource allocation policies for handling CPU time and memory allocation. The system follows a first-come-first-served (FCFS) scheduling policy for CPU allocation and a fixed partitioning scheme for memory allocation. Discuss the advantages and disadvantages of these policies in the context of a multi-user operating system. Suggest alternative policies that could potentially address the reported performance issues.

In the multi-user operating system UniOS, resource allocation is managed through a First-Come, First-Served (FCFS) scheduling policy for CPU allocation and a fixed partitioning scheme for memory allocation. While these policies offer simplicity and ease of implementation, they have advantages and disadvantages in a multi-user context. The FCFS scheduling policy is simple to implement as it processes tasks in the order they arrive without complex decision-making. It also ensures fairness since every process gets an equal opportunity to execute. However, it suffers from the convoy effect, where shorter processes experience significant delays if they are queued behind longer processes, leading to inefficient CPU utilization. Additionally, FCFS has poor average waiting time because if a lengthy process is at the front of the queue, subsequent processes must wait longer. Similarly, the fixed partitioning scheme for memory allocation has predictable memory management, as each partition is predefined, simplifying memory resource allocation. It also enhances isolation since processes remain confined within their respective partitions. However, it leads to internal fragmentation, where partitions remain partially unused if the allocated process does not require the entire partition size. Moreover, its limited flexibility means that processes requiring more memory than a single partition allows may be rejected or fail to execute (GeeksforGeeks, n.d.).

To address these challenges, UniOS could adopt alternative scheduling and memory allocation policies. One such alternative is the Round-Robin (RR) scheduling policy, which assigns a fixed time quantum to each process in the ready queue, cycling through them in a circular order. This approach enhances system responsiveness by ensuring all processes receive CPU time within a reasonable timeframe and promotes fairness by giving each process an equal share of CPU resources. However, RR introduces overhead due to frequent context switching, especially if the time quantum is too small, and selecting the appropriate time quantum is crucial to prevent performance degradation (Columbia University, n.d.). For memory management, UniOS could implement dynamic partitioning with paging, which allocates memory based on process requirements while dividing memory into fixed-size pages for non-contiguous allocation. This approach reduces fragmentation, as paging eliminates external fragmentation and minimizes internal fragmentation by allocating memory in fixed-size units. Additionally, it offers flexibility by allowing memory to be allocated as needed rather than being restricted to fixed partition sizes. However, dynamic partitioning with paging increases system complexity and introduces overhead due to page table management and page faults (Wikipedia, 2025). By implementing Round-Robin scheduling and dynamic partitioning with paging, UniOS can improve CPU utilization and memory efficiency, ultimately addressing its reported performance issues.

Deadlock Detection and Prevention

UniOS has encountered deadlock situations, leading to system crashes. Explain what a deadlock is in the context of an operating system. Discuss the strategies UniOS can implement to detect and prevent deadlocks. Provide specific examples of how these strategies can be applied in UniOS, considering the typical operations performed by users.

In operating systems, a deadlock occurs when a set of processes becomes permanently blocked because each process holds a resource while waiting for another resource that is held by another process, forming a cycle of dependencies that prevents any process from proceeding, leading to system halts or crashes (GeeksforGeeks, n.d.). To address deadlocks, UniOS can implement deadlock detection and prevention strategies. Deadlock detection allows the system to enter a deadlock state and then employs algorithms to identify it. Once detected, UniOS can recover by terminating one or more processes or preempting resources. A common approach is to periodically check for cycles in the resource allocation graph; if a cycle is detected, the system could terminate the lowest-priority process to break the deadlock (GeeksforGeeks, n.d.).

On the other hand, deadlock prevention ensures that deadlock conditions never occur by structurally eliminating at least one of the four necessary conditions for deadlock. The first condition, mutual exclusion, can be managed by making some resources sharable. However, in cases where mutual exclusion is unavoidable, such as printers in UniOS, only one process should be allowed access at a time. The hold and wait condition can be mitigated by requiring processes to request all required resources at once or release currently held resources before requesting new ones. For example, if a UniOS user needs both a scanner and a printer, they must request both simultaneously, reducing the likelihood of holding one resource while waiting for another. The no preemption strategy allows the system to forcibly take back resources from processes holding them if another process with a higher priority needs them. If a process in UniOS holds a file lock and subsequently requests access to the network but is denied, it would have to release the file lock and retry later. Finally, to eliminate circular wait, UniOS can impose an ordering of resource allocation and enforce those processes request resources in a predefined sequence. For instance, if the order is: (1) files, (2) printers, and (3) scanners, then a process requiring both a file and a printer must request the file first before the printer, thereby preventing circular dependencies (Javatpoint, n.d.). By implementing these detection and prevention strategies, UniOS can enhance system stability, reduce resource contention, and minimize the risk of system crashes due to deadlocks.

User Prioritization

UniOS currently treats all users equally in terms of resource allocation. Should UniOS prioritize certain users or user groups over others? If yes, what criteria should be used to determine the priority of users or groups? Discuss how implementing user prioritization might affect the overall system performance and user satisfaction.

In the current configuration of UniOS, all users receive equal treatment concerning resource allocation. However, implementing user prioritization could enhance system performance and user satisfaction. Prioritization ensures that critical tasks receive the necessary resources promptly, thereby improving efficiency. To determine user or group priorities, UniOS could consider criteria such as the urgency and impact of tasks, aligning with organizational goals. For instance, tasks that are high in importance and urgency should be prioritized to ensure that critical operations are completed promptly. Conversely, tasks with low importance and urgency can be scheduled for later, ensuring that resources are allocated where they are most needed (Javatpoint, n.d.).

Implementing user prioritization can significantly affect overall system performance and user satisfaction. By allocating resources based on task importance and urgency, the system ensures that critical operations are completed efficiently, leading to improved performance. Additionally, users responsible for high-priority tasks will experience reduced wait times and increased responsiveness, enhancing their satisfaction. However, it's essential to manage this approach carefully to prevent lower-priority users from experiencing significant delays, which could lead to dissatisfaction. Regularly reviewing and adjusting prioritization criteria can help maintain a balance, ensuring that all users receive adequate resources while aligning with organizational objectives (Javatpoint, n.d.).

System Monitoring and Performance Tuning

To address the reported performance issues, UniOS plans to implement a system monitoring tool that collects data on CPU and memory usage, I/O operations, and response times. Describe the key metrics and data points that the monitoring tool should collect. Additionally, outline the steps the UniOS administrators can take based on the collected data to tune the system's performance.

To address the reported performance issues, UniOS plans to implement a system monitoring tool that collects data on key performance metrics such as CPU and memory usage, I/O operations, and response times. The monitoring tool should track several critical metrics, including CPU utilization, which measures the percentage of CPU capacity in use and helps identify excessive workloads that may cause system slowdowns or crashes. Memory usage should also be monitored to detect high RAM consumption, which can degrade performance or lead to application failures due to memory leaks. Additionally, disk usage is a vital metric, as excessive disk consumption can create bottlenecks and negatively impact storage availability and performance. Network traffic analysis is essential to detect potential bandwidth issues or security threats, such as Distributed Denial of Service (DDoS) attacks, that may impair UniOS operations. Another crucial metric is server response time, which assesses how quickly UniOS responds to user requests, as increased response times can indicate performance bottlenecks requiring optimization. Error rates should also be tracked, as frequent errors can signal software bugs, misconfigurations, or hardware malfunctions that must be addressed promptly (Kaseya, n.d.). Furthermore, monitoring system load, which evaluates the total workload across CPU, memory, and disk resources, allows administrators to optimize load distribution. Lastly, uptime and downtime metrics ensure that UniOS meets service level agreements (SLAs) by measuring system reliability and identifying periods of unavailability (Kaseya, n.d.).

Based on the collected data, UniOS administrators can take several performance tuning steps. First, setting performance baselines helps establish expected system behavior, allowing deviations to be quickly identified and addressed. Comprehensive monitoring should be employed to observe all critical components, ensuring that resource bottlenecks and inefficiencies are detected early. When necessary, upgrading server hardware can improve performance, particularly in cases where CPU, memory, or storage limitations impact system efficiency (Eyer, n.d.). Additionally, optimizing operating system settings, such as process scheduling and memory management, can enhance resource utilization and responsiveness (Linux Journal, n.d.). Enhancing network performance by optimizing bandwidth allocation and network configuration reduces latency and improves throughput, supporting faster data transmission (Eyer, n.d.). Load balancing should be implemented to distribute tasks across multiple resources, preventing any single server from becoming a bottleneck and ensuring better system stability. Database performance tuning, which includes optimizing queries and indexing, is another critical step that can significantly reduce processing time and resource consumption (Eyer, n.d.). To further enhance performance, caching mechanisms should be employed to store frequently accessed data, reducing the need for repeated data retrieval operations. Regular performance audits are necessary to analyze system logs, detect anomalies, and proactively resolve emerging issues before they affect user experience. Finally, capacity planning based on historical performance data enables UniOS to scale resources in anticipation of future demand, preventing system degradation as workloads increase (Eyer, n.d.). By systematically monitoring these key metrics and implementing targeted performance optimizations, UniOS can enhance system responsiveness, reliability, and overall user satisfaction.

User Education and Communication

To mitigate resource allocation and deadlock-related problems, UniOS plans to educate its users about best practices and potential issues. What communication channels should UniOS use to reach its users effectively? Provide examples of messages or guidelines that UniOS can communicate to users to help them use the system efficiently and responsibly.

To effectively educate UniOS users on best practices for resource allocation and deadlock prevention, it is crucial to leverage diverse communication channels that cater to different user preferences. A combination of email newsletters, instant messaging platforms, video conferencing, project management platforms, and internal social media/intranet can enhance outreach and engagement. Email newsletters can provide detailed guidelines, updates, and best practices, ensuring that users receive structured information; however, excessive emails should be avoided to prevent disengagement (SchoolStatus, n.d.). Instant messaging platforms such as Slack or Microsoft Teams allow for real-time communication, ensuring that updates, reminders, and discussions are promptly shared (Prezent.ai, n.d.). Video conferencing tools like Zoom can be used for live training sessions, interactive demonstrations, and Q&A sessions, fostering better user understanding and engagement (Prezent.ai, n.d.). Additionally, project management platforms like Trello or Asana can serve as repositories for educational resources, ensuring structured access to guidelines and tracking compliance (Prezent.ai, n.d.). Lastly, an internal social media or intranet platform can create a collaborative space where users discuss best practices, share experiences, and access training materials, encouraging peer learning and community building (GetTalkative, n.d.). By implementing these communication methods, UniOS can ensure that information on resource allocation and deadlock prevention reaches its users effectively.

In terms of content, UniOS should provide actionable guidelines to help users operate the system efficiently and responsibly. For resource allocation, users should be encouraged to request only the resources they need, as unnecessary resource hoarding can lead to inefficiencies and system slowdowns. Additionally, users should be advised to release resources as soon as they are no longer needed to improve system availability. Regarding deadlock prevention, UniOS can educate users on requesting resources in a specific order to avoid circular waits, which are a primary cause of deadlocks (GeeksforGeeks, n.d.). Users should also be informed about UniOS's timeout mechanisms, which limit the duration a process can wait for a resource, thus preventing indefinite waiting times (GeeksforGeeks, n.d.). To enhance efficient system usage, UniOS should recommend scheduling resource-intensive tasks during off-peak hours to balance the system load and optimize performance. Additionally, providing users with monitoring tools within UniOS can help them track their resource consumption, allowing them to identify inefficiencies and make necessary adjustments. By distributing these best practices through multiple communication channels, UniOS can empower its users to manage system resources more effectively, thereby reducing deadlock occurrences and improving overall system efficiency and user satisfaction.

Reference:

The 8 most effective communication channels in business. RSS. (n.d.). https://www.prezent.ai/zenpedia/communication-channels

Deadlock detection and recovery. GeeksforGeeks. (2025a, January 29). https://www.geeksforgeeks.org/deadlock-detection-recovery/

Deadlock prevention and avoidance. GeeksforGeeks. (2025b, January 29). https://www.geeksforgeeks.org/deadlock-prevention/

Deadlock prevention in OS (operating system) - javatpoint. www.javatpoint.com. (n.d.). https://www.javatpoint.com/os-deadlock-prevention

FCFS - first come first serve CPU scheduling. GeeksforGeeks. (2025c, January 21). https://www.geeksforgeeks.org/first-come-first-serve-cpu-scheduling-non-preemptive/

Introduction of deadlock in operating system. GeeksforGeeks. (2025d, January 16). https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/

Key server monitoring metrics for measuring performance. Kaseya. (2024, September 4). https://www.kaseya.com/blog/server-monitoring-metrics/

Project prioritization criteria: Step-by-step guide. OnePlan. (2024, December 17). https://oneplan.ai/articles/prioritization-criteria-step-by-step-guide/

SchoolStatus. (2024, November 19). *The 6 School Digital Communication Channels Overview*. https://www.schoolstatus.com/blog/6-key-school-communication-channels-and-how-to-use-them

Server performance monitoring & tuning: 10 best practices. Eyer. (2024, July 4). https://eyer.ai/blog/server-performance-monitoring-and-tuning-10-best-practices/

Talkative. (2025, February 4). *The 10 most essential digital communication channels for business*. Video Chat, Live Chat, Chatbot, Cobrowsing Software. https://gettalkative.com/info/communication-channels

Whittaker, G. (n.d.). *#linux*. Home. https://www.linuxjournal.com/content/system-performance-monitoring-and-tuning-guide

Wikimedia Foundation. (2024, September 10). *Memory management (operating systems)*. Wikipedia. https://en.wikipedia.org/wiki/Memory_management_%28operating_systems%29

Yang, J. (n.d.). Operating System - Scheduling. https://www.cs.columbia.edu/~junfeng/12sp-w4118/lectures/l12-sched.pdf