**Technological Institute of the Philippines**

**938 Aurora Blvd. Cubao, Quezon City**

**College of Computer Studies**

**CS 007 - Parallel and Distributed Computing**

**Prelim Period**

| Name: Tiamzon, Bryan Dominick A. | Date: 06/03/2024 |
| --- | --- |
| Program / Section: BSCS / CS33S1 | Instructor: Ma’am Janice Capule |
| Instructions: Perform the tasks below. (20 points)  1. Create a table that will compare serial computing from parallel computing. (10)   |  | **Serial Computing** | **Parallel Computing** | | --- | --- | --- | | Advantages | * Simpler to design, debug, and implement. * Ideal for activities that demand sequential execution. * There is no need to manage communication and synchronization between CPUs. * Produces dependable outcomes without concurrency concerns. | * Can handle massive computations more quickly by spreading jobs across several processors. * Suitable for large-scale challenges by adding additional CPUs. * Effectively employs multi-core computers and distributed systems. * Parallelizable jobs need significantly less computing time. | | Disadvantages | * A single processor's clock speed limits performance. * Not suitable for large, sophisticated issues that need significant processing capacity. * Underutilized multi-core CPUs accessible in current gear. | * More difficult to develop, implement, and debug. * Managing communication and synchronization can increase overhead. * Deadlocks and racing circumstances must be handled carefully. * Performance increases are determined by the nature of the issue and its design. | | Algorithm | * Executes instructions sequentially, from beginning to end. | * Divides a challenge into smaller jobs that may be addressed concurrently. | | Implementation | * RS-232 serial interface transmits data bits consecutively. | * Parallel Algorithm Scheduling Library (PASL) for shared memory parallelism. |   2.Create a table that will compare each process scheduling algorithm. (10)   |  | **Advantages** | **Disadvantages** | | --- | --- | --- | | First Come First Serve | * Simple to comprehend and apply. * Processes are executed in the sequence in which they come. * Every procedure has its turn. | * Long procedures can cause shorter ones to take longer to complete. * If a long process comes before a shorter one, inefficiencies may arise. * Interactive users experience a lack of reactivity. | | Shortest-Job-First | * Reduces the average waiting time. * Suitable for jobs with predetermined execution times. | * Longer processes may suffer from indefinite delay. * Accurate task length estimate is frequently not achievable. * Lacks reactivity. | | Shortest Remaining Time | * Dynamic version of SJF that preempts longer processes for shorter ones. * Suitable for systems where new short processes frequently arrive. | * Longer processes can be continuously preempted. * High context switching overhead due to frequent preemption. * Similar to SJF, predicting job lengths accurately is challenging | | Priority Scheduling | * Can prioritize crucial jobs over less important ones. * High-priority jobs receive resources swiftly. | * Low-priority processes may never run. * Managing and allocating priorities may be difficult. * This may result in unexpected wait times for low-priority tasks. | | Round Robin Scheduling | * Each process receives an equal proportion of CPU time. * Suitable for time-sharing systems with interactive users. * Ensures that all processes are carried out. | * Frequent context shifts might reduce performance. * Choosing the appropriate quantum is crucial; too tiny results in overhead, while too high results in FCFS-like behavior. * This can lead to longer average waiting times. | | Multilevel Queue Scheduling | * Different queues for different kinds of operations. * Higher priority queues can be allocated more CPU time. | * Processes are permanently allocated to a queue, which may be inefficient over time. * Lower-priority queues may experience hunger. * Managing many queues and rules adds complexity. | | Multilevel Feedback Queue Scheduling | * Processes can transition across queues according to their behavior and needs. * Combines the benefits of many scheduling strategies. * Processes gradually receive CPU time when they migrate to lower priority queues. | * Implementation and management are complicated. * Frequent changes and context transitions might generate overhead. * To get best performance, parameters and regulations must be carefully tuned. |   **Individual synthesis**  A comparison of serial versus parallel computing shows that, while serial computing is simpler and more predictable, parallel computing provides considerable performance and scalability improvements. However, the complexity and overhead associated with parallel processing must be properly handled. Each algorithm has distinct benefits and problems in process scheduling, ranging from the simplicity of FCFS to the versatility of multi-layer feedback queues. To enhance performance and resource consumption, the scheduling algorithm used must strike a balance between efficiency, fairness, and system needs.  **References:**   * https://massive.io/file-transfer/serial-vs-parallel-write-speed/ * https://www.hp.com/us-en/shop/tech-takes/parallel-computing-and-its-modern-uses * https://www.geeksforgeeks.org/difference-between-sequential-and-parallel-computing/ * https://www.geeksforgeeks.org/advantages-and-disadvantages-of-various-cpu-scheduling-algorithms/ * https://www.studytonight.com/operating-system/comparision-scheduling-algorithms   **Honor Pledge for Graded Assignments**  *“I affirm that I have not given or received any unauthorized help on this assignment and that all*  *work shall be my own.”* | |