**OS Assignment No 4**

**Name**: Affan Ahmad BS(cs) 5-A

**Enroll**: 03-134221-003

**Question NO 1:**

For simulating CPU scheduling schemes and memory management techniques visually, we can use various programming languages along with graphical libraries. Here are some relevant simulation tools for each category:

1. **Python with Matplotlib for CPU Scheduling:**

**Resources Utilization:**

Matplotlib allows for the creation of graphical representations of CPU scheduling algorithms, including processes, CPU bursts, and scheduling queues. While Matplotlib itself is lightweight, the visualization may consume additional memory and processing power, especially for complex simulations.

**Performance:**

Matplotlib provides a high-level interface for creating visualizations, which can be efficient for small to medium-sized simulations. However, for large-scale simulations with a significant amount of data, the performance may degrade, leading to slower rendering times and decreased interactivity.

**Efficiency:**

Matplotlib is efficient in terms of ease of use and flexibility for creating visualizations of CPU scheduling algorithms. However, the efficiency of the simulation itself depends on the underlying Python code and algorithm implementation.

1. **Java with JavaFX or Swing for CPU Scheduling:**

**Resources Utilization:**

JavaFX and Swing provide robust tools for creating interactive visualizations of CPU scheduling algorithms. These libraries may consume more system resources compared to Matplotlib due to their extensive GUI capabilities.

**Performance:**

JavaFX and Swing offer excellent performance for creating complex GUI applications, including visual simulations. They can handle large-scale simulations with ease and provide smooth rendering and user interaction.

**Efficiency:**

JavaFX and Swing are efficient in terms of performance and provide a wide range of features for creating interactive visualizations of CPU scheduling algorithms. However, the learning curve for Java and GUI development may affect efficiency during initial development stages.

1. **Python with Pygame for Memory Management:**

**Resources Utilization:** Pygame is designed for writing video games and provides a wide range of multimedia capabilities. While it may consume more resources compared to Matplotlib, it offers more advanced graphical features suitable for visual simulations of memory management techniques.

**Performance:** Pygame offers good performance for creating visual simulations, especially for real-time interactions and animations. It can handle complex simulations with ease and provides smooth rendering even for large-scale simulations.

**Efficiency:** Pygame is efficient for creating visual simulations of memory management techniques, thanks to its comprehensive multimedia capabilities. It allows for detailed representations of physical and virtual memory structures, page tables, and memory allocation algorithms.

1. **C++ with OpenGL or SFML for Memory Management and Virtual Memory:**

**Resources Utilization:**

OpenGL and SFML provide low-level access to graphics hardware, allowing for detailed visualizations of memory management and virtual memory techniques. While they offer high performance, they may require more memory and processing power compared to Python-based solutions.

**Performance:**

OpenGL and SFML offer excellent performance for creating graphical simulations, especially for real-time rendering and interactive applications. They can handle complex simulations efficiently and provide smooth animations and user interactions.

**Efficiency:** OpenGL and SFML are efficient in terms of performance and resource utilization, making them suitable for creating detailed visualizations of memory management and virtual memory techniques. However, they may have a steeper learning curve compared to Python-based solutions, especially for beginners.

1. **Round Robin CPU Scheduling:**

**Resources Utilization:**

Round Robin ensures fair utilization of CPU resources among all processes by allocating each process a fixed time slice. This prevents any single process from monopolizing the CPU. However, if the time quantum is too long, CPU resources may be underutilized.

**Performance:**

Round Robin provides reasonable performance for interactive systems as it guarantees a response time within a certain bound. However, for long CPU-bound processes, it may lead to high context switch overhead and decreased throughput.

**Efficiency:**

Round Robin is efficient in terms of simplicity and fairness. However, the choice of time quantum significantly affects its efficiency. Too short a time quantum may lead to frequent context switches, while too long a time quantum may degrade system responsiveness.

1. **First Come First Serve (FCFS) CPU Scheduling:**

**Resources Utilization:**

FCFS utilizes CPU resources by executing processes in the order they arrive. It may lead to poor utilization if long CPU-bound processes arrive first, causing short CPU-bound processes to wait excessively.

**Performance:**

FCFS may result in poor performance for interactive systems as it doesn't prioritize short jobs or guarantee a bounded response time. Long CPU-bound processes may monopolize the CPU, leading to increased response times for other processes.

**Efficiency:**

FCFS is simple to implement and understand but may not be efficient in terms of overall system performance and responsiveness, especially in scenarios with a mix of short and long processes.