ASSIGNMENT

Compare any 3 operating system for the following services:

a. Process Management
b. Multithreading
c. CPU scheduling.

→ Process Management
↳ LINUX

Model: Processes in Linux are isolated execution units with thier own address space, file discriptions and environment variables. The kernal uses structures like task_struct to manage processes, allowing features like parent child relationships and signals for inter-process communication.

Scheduler: Linux's completly fair scheduler (CFS) keeps track of each process's virtual runtime and tries to ensure proportional CPU allocation. Processes are organized in a red-black tree structure to efficiently determine which process should run next.

Control & Tools: Commands like ps, top, htop, and kill allow detailed process control. nice and renice let users influence priority levels, while cgroups (control groups) offer fine-grained resource allocation by grouping processes.

↳ WINDOWS

Model: Window uses the NT kernal's process model, where each process has a unique process control block (PCB), it's own virtual address space, and handles for objects like files and events. The process archi--tecture supports threads as lightweight execution unit within processes.

Scheduler: Windows employs a priority-driven premetive scheduling system with real-time and base priority classes. It dynamically adjusts priorities based on CPU usage and interactivity, giving preference to foreground applications for better responses.

Control & Tools : The task manager provides a graphical interface for process inspection, while APIs like OpenProcess allow programmatic process manipulation. Power users can leverage wmic and Powershell commands for automation.

↳ macOS

Model : macOS is built on the XNU kernal, which integrates Mach and BSD components. Each process has it's own memory space and can create threads managed by kernal.

Scheduler : macOS's scheduler is a hybrid model combining preemitive multitasking with cooperative elements in certain contexts. It adapts to workloads, ensuring smooth UI responsiveness while balancing CPU resources accross processes.

Control & Tools : User can monitor processes via top, ps, and Activity Monitor. Developers can manipulate processes through Mach APIs, BSD commands, or higher-level frameworks like Grand Central Dispatch (GCD), which abstracts thread & process management.

→ Multithreading

↳ LINUX

Model : linux supports kernal threads as well as user threads implemented via the Native POSIX Thread Library (NPTL). Treads share the process's memory space, open files, and other resources but have individual stack, registers and thread IDs.

Implementation : The clone () system call allows prosesses to share resources like memory and file descriptions selectivly, enabling light weight threads. Thread scheduling integrates with the CFS for fairness.

Control & Tools : The pthread library provides functions such as pthread_create, pthread_join, pthread_mutex_lock, allowing fine control over thread creation, synchronization, and communication. Debugging is supported via gdb and other tools.

↳ WINDOWS

Model: In windows, threads are treated as the primary execution unit within processes. Each thread has it's own stack, CPU context, and priority but shares the process's virtual memory and handles.

Implementation: Threads are created using the CreateThread or _beginthreadx functions. The kernal schedules threads based on priority classes and quantum times, allowing applications to specify real-time or background scheduling preferences.

Control & Tools: The windows API includes thread manipulation functions ~~and~~ such as SuspendThread, ResumeThread, SetThread-Priority, and WaitForSingleObject. Performance monitoring tools and debuggers allow tracking of thread states, resources consumption, and deadlocks.

↳ macOS

Model : macOS threads are kernal-managed, and processes can spawn multiple threads sharing resources like memory and file descriptions. Apple provides higher-level abstractions for concurrency to simplify multithreading.

Implementation : In addition to POSIX ~~to~~ threads (pthread), macOS emphasizes Grand Central Dispatch (GCD) and operation Queues to handle thread pools, task scheduling, and load balancing across cores. This frameworks dynamically optimizes resources utilization.

Control & Tools : Developers can use pthread for low level threading or leverage GCD for higher-level constructs such as dispatch queues, barriers and semaphores. Instruments and Activity Monitor help trace thread contention, CPU spikes, and deadlocks.

→ CPU Scheduling

↳ LINUX

<u>Algorithm</u>: The completly fair scheduler (CFS) ensures that processes are allocated CPU time proportionally based on their weight and runtime history. It prevents starvation by tracking virtual runtime and balancing fairness and efficiency.

<u>Features</u>: Linux allows setting CPU affinity using taskset, enabling processes to run on specific cores, which is useful for performance tuning. Real-time scheduling policies (SCHED-RR) are available for time-sensitive tasks.

<u>Control & Tools</u>: Users can adjust priorities with nice and renice, or directly manipulate scheduling policies using chrt. System monitoring via htop displays CPU usage by process or thread, while pref offers advanced profiling.

↳ WINDOWS

Algorithm: Windows uses a priority class, premitive scheduler with a combination of real-time and variable prio-rity classes. It dynamically boosts priorities to prevent UI sluggishness and balances between CPU-bound and I/O-bound processes.

Features: The system distinguishes b/w base priority and dynamic priority, allowing applications to request more or less CPU time. Windows also optimizes scheduling based on process state transitions, like waiting for I/O.

Control & Tools: SetThreadPriority and SetProcessPriority let developers influence scheduling, while Task manager provides a graphical overview of CPU load and active processes. Performance counters (perfmon) and PowerShell scripts provide deeper insights.

L) mac OS

Algorithm: macOS employs a hybrid scheduler combining preemptive multitasking for general workloads and cooperative adjustments in UI-intensive or multimedia contexts. It optimizes for responsiveness and energy efficiency.

Features: CPU scheduling integrates with system-wide power management, intelligently scaling CPU frequency ~~at~~ and adjusting task priorities based on workload patterns, especially on Apple's M-series chips.

Control & Tools: Users can adjust process priority using renice, while developers can fine-tune dispatch queues in GCD. Instruments offers detailed CPU activity traces, and Activity Monitor gives real-time data on thread utilization.