TEORI TUGAS 2 SISTEM OPERASI

PRACTICE EXERCISE

Nama : Ferry Ferdiansyah

NRP : 3124500050

Dosen Pengajar : Dr Ferry Astika Saputra ST, M.Sc

PROGRAM STUDI D3 TEKNIK INFORMATIKA POLITEKNIK ELEKTRONIKA NEGERI SURABAYA (PENS)

TAHUN 2025

1. What are the three main purposes of an operating system?
   - Resource Management:
     The OS manages hardware resources such as the CPU, memory, storage, and peripheral devices. It ensures efficient allocation and utilization of these resources among multiple applications and users.
   - Abstraction and Simplification:
     The OS provides a simplified interface for users and applications to interact with the hardware. It hides the complexity of hardware operations through abstractions like file systems, device drivers, and system calls.
   - System Protection and Security:
     The OS ensures that different programs and users do not interfere with each other. It enforces security policies, manages user permissions, and protects the system from unauthorized access or malicious activities.
2. When is it appropriate for the operating system to forsake efficiency and "waste" resources? Why is such a system not really wasteful?
   It is appropriate for the operating system to forsake efficiency and "waste" resources in scenarios where improving reliability, security, or user experience takes priority over raw performance. While this may seem wasteful, such trade-offs are often necessary to achieve higher-level goals. Here are some examples and reasons why this is not truly wasteful:
   a. Redundancy for Reliability:
      - The OS may allocate extra resources (e.g., memory, storage, or CPU cycles) to create backups, redundancy, or fail-safes. For example, keeping multiple copies of critical data or running background checksums to detect errors.
      - Why it's not wasteful: This ensures system stability and data integrity, preventing costly failures or data loss.
   b. Security Measures:
      - The OS might use additional resources to enforce security policies, such as encryption, access control, or sandboxing applications. These measures can slow down performance but are critical for protecting the system.
      - Why it's not wasteful: The cost of a security breach (e.g., data theft, malware infection) far outweighs the resource overhead.

c. User Experience and Responsiveness:
   - The OS may prioritize responsiveness over efficiency, such as preloading frequently used applications into memory or keeping idle processes in RAM for quick access. This can appear wasteful but improves the user experience.
   - Why it's not wasteful: A faster, more responsive system enhances productivity and user satisfaction, which justifies the resource usage.

d. Future-Proofing and Scalability:
   - The OS might reserve resources for future tasks or scalability, such as keeping some memory or CPU capacity free to handle sudden spikes in demand.
   - Why it's not wasteful: This ensures the system can handle unexpected workloads without crashing or slowing down.

3. What is the main difficulty that a programmer must overcome in writing an operating system for a real-time environment?

The main difficulty a programmer must overcome when writing an operating system (OS) for a real-time environment is ensuring deterministic timing and predictability. In real-time systems, tasks must be completed within strict deadlines, and failure to meet these deadlines can lead to system failure or catastrophic consequences. Here are the key challenges associated with this:

a. Guaranteeing Timely Execution:
   - Real-time systems require that tasks are executed within precise time constraints. The OS must ensure that high-priority tasks are always completed on time, even in the presence of lower-priority tasks or system overhead.
   - Challenge: Designing scheduling algorithms that prioritize tasks based on their deadlines and guarantee timely execution.

b. Minimizing Latency and Jitter:
   - Real-time systems must have minimal and predictable response times (latency) and avoid variability in task execution times (jitter).
   - Challenge: Reducing interrupt handling times, context switching overhead, and other sources of delay to ensure consistent performance.

c. Resource Management Under Constraints:

- Real-time systems often operate with limited resources (e.g., memory, CPU, power). The OS must manage these resources efficiently while still meeting timing requirements.
- Challenge: Balancing resource allocation to ensure that critical tasks always have the resources they need without over-provisioning.

d. Handling Concurrency and Synchronization:
   - Real-time systems often involve multiple tasks running concurrently, which can lead to race conditions or deadlocks if not managed properly.
   - Challenge: Implementing synchronization mechanisms (e.g., semaphores, mutexes) that do not introduce unpredictable delays.

e. Predictable Behavior in All Scenarios:
   - The OS must behave predictably even under worst-case conditions, such as high load or hardware failures.
   - Challenge: Thorough testing and validation to ensure the system meets its timing requirements in all possible scenarios.

f. Integration with Hardware:
   - Real-time systems often interact directly with hardware, requiring precise control over hardware resources.
   - Challenge: Writing low-level code that interacts with hardware while maintaining real-time guarantees.

4. Should the operating system include applications such as web browsers and mail programs? Argue both sides.

   Yes, it should include them:

   a. Convenience: Pre-installed applications provide a ready-to-use system for users.
   b. Integration: Applications can be tightly integrated with the OS for better performance and security.
   c. Consistency: Ensures a uniform user experience across devices.

   No, it should not include them:

   a. Bloatware: Including unnecessary applications can bloat the OS, consuming resources.
   b. Flexibility: Users may prefer alternative applications, and pre-installed apps limit choice.
   c. Security Risks: More applications increase the attack surface for potential vulnerabilities.

5. How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security)?
    a. Kernel Mode: The OS runs in kernel mode, allowing direct access to hardware and critical system resources.
    b. User Mode: Applications run in user mode, with restricted access to hardware and sensitive areas of the system.

    This separation prevents user applications from interfering with the OS or other applications, ensuring system stability and security.
6. Which of the following instructions should be privileged?
    Privileged instructions are those that can only be executed in kernel mode to protect system integrity. The privileged instructions are:
    a. Set value of timer.
    b. Clear memory.
    c. Turn off interrupts.
    d. Modify entries in device-status table.
    e. Switch from user to kernel mode.
    f. Access I/O device.

    Non-privileged instructions:
    a. Read the clock.
    b. Issue a trap instruction.
7. Describe two difficulties that could arise from placing the OS in a memory partition that cannot be modified.
    a. Limited Flexibility: The OS cannot dynamically adjust its memory usage, making it difficult to handle varying workloads or new features.
    b. Performance Bottlenecks: If the OS cannot modify its own memory, it may struggle to optimize performance or respond to system events efficiently.
8. What are two possible uses of multiple modes of operation in CPUs?
    a. Enhanced Security: Additional modes can provide finer-grained access control, isolating critical processes from less trusted ones.
    b. Specialized Operations: Modes can be designed for specific tasks, such as virtualization or real-time processing, improving efficiency.
9. How could timers be used to compute the current time?
    Timers can be used to track elapsed time by counting clock ticks. For example:

a. A system timer increments at a fixed frequency (e.g., 1,000 ticks per second).
b. The OS records the number of ticks since a known start time (e.g., system boot).
c. By multiplying the tick count by the interval between ticks, the OS can calculate the current time.

10. Give two reasons why caches are useful. What problems do they solve? What problems do they cause?

Reasons caches are useful:

a. Speed: Caches provide faster access to frequently used data, reducing latency.
b. Efficiency: They reduce the load on slower storage devices (e.g., disks) by serving data from faster memory.

Problems they solve:

a. Slow access times for data stored in main memory or secondary storage.
b. High latency in fetching data from remote or slow devices.

Problems they cause:

a. Cache Coherence: Ensuring consistency between cache and main memory.
b. Cache Pollution: Storing unnecessary data, reducing effective cache size.

Why not make caches as large as the device they cache?

a. Cost: Larger caches are more expensive.
b. Diminishing Returns: Beyond a certain size, the performance gains do not justify the cost.