# OS CW2

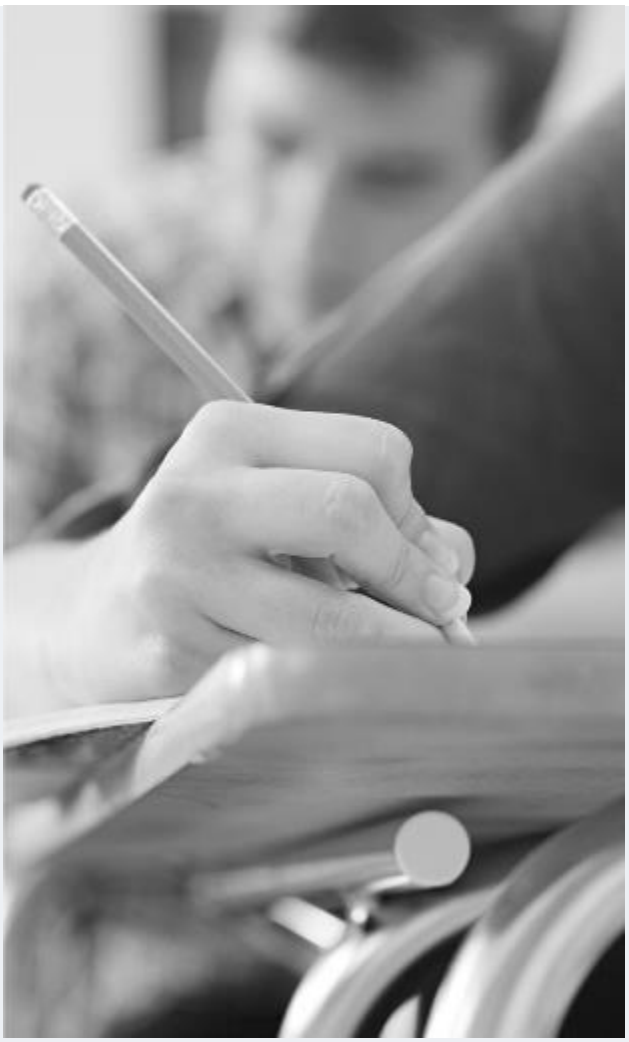
*4/15/2023*
*Operating system 2*

**Multi-process environment**

*A computing system or software architecture that supports several processes running concurrently and independently is known as a multi-process environment. The scheduling and execution of these activities in such a setting are controlled by the operating system or a specialized software framework. In a multi-process environment, each process has its own execution context and memory space, enabling it to carry out tasks on its own. Through different inter-process communication protocols offered by the operating system or framework, these processes can communicate with one another.*

*Processes can be started, stopped, and scheduled to execute on one or more CPUs or processing cores in a multi-process environment. This parallel execution enables more efficient use of system resources and can enhance system responsiveness and performance.*

*Operating systems, servers, and distributed systems frequently employ multi-process environments to manage numerous tasks at once. Building scalable and robust systems as well as dealing with computationally demanding or time-consuming activities make use of them in particular. Multi-process setups further offer a degree of separation between processes, improving system stability and security.*



**Concurrent processes**

*Multithreading is a method of running concurrent processes by employing numerous threads. Multiple threads are established within a single process in a multithreaded environment, and each thread is capable of carrying out a different course of execution concurrently. The fact that these threads share the same process resources and memory space enables effective data exchange and communication.*

1. *Multiple threads can be formed inside a process by utilizing threading libraries or language-specific techniques. Each thread stands for a distinct execution route.*
2. *Operating systems or thread schedulers control how threads are scheduled onto available CPU cores or processors. Each thread is given a time slice by the scheduler, which enables concurrent or interleaved execution.*
3. *Resources that are shared: Since threads access and alter shared data since they share the same process's memory and resources. To prevent conflicts and data inconsistencies, strict synchronization is necessary with this shared access. To coordinate access to shared resources, synchronization methods like locks, semaphores, and mutexes are utilized.*
4. *Threads can interact and collaborate with one another inside a process using shared memory or inter-thread communication techniques like message forwarding or event signaling. These techniques enable data sharing, job completion signaling, and task execution coordination amongst threads.*
5. *Threads are capable of finishing their execution independently of one another. A thread can either terminate or be connected with additional threads once it has completed its assigned work or encountered a termination circumstance. The main thread or other threads can wait for a thread's conclusion so they can access its results by joining it.*

## Sharing Data in a multithreaded environment

In a multi-threaded system, threads share memory to move data among each other. Because they share the same memory space, threads inside a process can instantaneously read from and write to shared variables, data structures, and resources.

Here are some examples of how threads often communicate data:

1. 1. Shared variables: Threads may access and modify shared variables that are stored in the same memory location. When several threads access and modify the same variable at once, synchronisation mechanisms must be employed to avoid data races and ensure data consistency. Locks, mutexes, and atomic operations are examples of synchronisation primitives that are used to coordinate access to shared variables.

2. 2. Threads can programme shared data structures like queues, stacks, arrays, and linked lists. Conflicts occur when several threads access or modify the same data structure at the same time; hence, adequate synchronisation is necessary to prevent them. Synchronisation methods, such as locks or concurrent data structures designed for thread safety, can be used to ensure data integrity.

3. Using message forwarding techniques, thread communication and data sharing are made feasible. Instead of directly accessing shared memory, threads can interact with one another by passing messages containing data to other threads. With this approach, explicit synchronisation is not necessary, and data isolation is ensured. To achieve message passing, a variety of techniques can be employed, such as pipes, queues, channels, or message passing libraries.

4. Thread-local storage (TLS): Threads can have a TLS of their own to store thread-specific information in addition to shared data. TLS allows for the possibility of each thread having its own private data that is not shared with other threads. This is crucial for maintaining thread-specific state or lowering synchronization costs when specialized data is only needed within a certain thread.

To avoid data races, inconsistent state, and other concurrency-related problems, exchanging data among threads necessitates proper synchronization. When many threads interact with common data, synchronization methods like locks, semaphores, or atomic operations must be used to coordinate access and guarantee thread safety.

When two or more threads visit a shared variable simultaneously without sufficient synchronization and at least one of the accesses involves a write operation, it is known as a data race. Data races cause unpredictable behavior and might have unintended and incorrect program results.

Inconsistent state usually happens when many threads simultaneously access and modify shared data without enough synchronisation in a concurrent or multi-threaded setting. If there is no coordination, numerous threads working on the same piece of data may clash and provide erroneous results.

- • Synchronisation primitives called locks are employed to guarantee mutual exclusion. Oftentimes, locks are referred to as mutexes, which is short for mutual exclusion. A lock, which grants exclusive

access to a shared resource, may only be held by one thread at one. When a thread needs to access the shared data, it queries the lock. If the lock is being held by another thread, the requesting thread will be halted until it is released. As a result, since only one thread is ever able to access the shared data, concurrent modifications that may cause data races are avoided.

- • Semaphores are synchronisation techniques that allow a limited number of concurrent threads to access a shared resource. The quantity of available resources is counted using a semaphore. When a thread needs to access a shared resource, it checks the semaphore. When the count rises over 0, the thread lowers the count and keeps going. If the count is zero, indicating that all resources are currently in use, the thread is suspended until a resource becomes available. Semaphores can be used to control access to a limited amount of resources and concurrent execution.

- Atomic operations are those that are carried out alone, without interference from other threads. These procedures enable the read-modify-write cycle to be atomic and irreversible. Atomic operations, which are frequently provided by the programming language or hardware, make sure that the operation is either completely completed or not undertaken at all. They enable simple operations to be carried out without the need for explicit locks or semaphores, such as incrementing or decrementing shared variables. Atomic operations help prevent data races by assuring that concurrent edits to shared data do not produce inconsistent results.

By using locks, semaphores, and atomic actions, developers may impose accurate synchronisation and coordinate access to shared data. These techniques help to ensure thread safety, avoid data races, and guarantee the consistency and integrity of shared data in a concurrent situation.

### How do threads communicate in a multithreaded environment

In a multithreaded system, threads can interact with one another utilising a number of strategies that make it simpler to share information and plan operations. Here are a few frequently utilised thread communication techniques:

1. Shared Memory: All of a process's threads have access to its memory. Common resources, data structures, and variables are under their direct control and access. Proper synchronisation mechanisms, such locks or atomic operations, are used to synchronise access to shared memory and prevent data races or inconsistencies.

2. Message passing enables threads to communicate by having one transmit a message containing data or instructions to another. Message passing may be achieved using a variety of techniques, including queues, pipelines, channels, and message passing libraries. Synchronisation may be required to ensure proper communications ordering and synchronisation between threads.

3. By using condition variables, which are synchronisation primitives, threads can wait until a certain condition is satisfied before moving forward. Threads may stall on a condition variable until another thread signals that the condition has been satisfied. In order to accomplish thread coordination patterns like reader-writer or producer-consumer, locks and condition variables are typically utilised.

4. Synchronisation primitives that may be used to synchronise and coordinate the activity of several threads include locks, mutexes, semaphores, and barriers. These restrictions provide mutual exclusion, govern access to shared resources, and establish synchronisation points for thread coordination.

5. Thread signalling: Threads can signal or alert one another when a certain event or condition change takes place. It is possible to signal using flags, condition variables, or specific signalling techniques provided by the programming language or threading library. Signalling facilitates thread coordination and synchronisation by allowing threads to react to changes or perform the necessary activity.

6. Thread affiliation: Threads can wait for other threads to complete their runs by affiliating with them. Before joining another thread, a thread waits for the connected thread to finish running. This makes thread dependencies or sequential execution possible, ensuring that certain actions are carried out in the right order.

The communication method to be utilised is determined by the architecture of the multithreaded programme and its particular requirements. When designing thread communication, developers must carefully consider thread safety, synchronisation, and potential bottlenecks in order to ensure accurate and efficient concurrent behaviour.

## Security features

In multi-user and multi-process contexts, operating systems offer a variety of security services to guarantee a safe environment and protect data integrity. Operating systems frequently have access control, authentication, and authorization security capabilities. Let's talk about each of these attributes and how they enhance security.

1. Access Control:

An essential security function, access control limits and regulates access to resources based on user rights and permissions. It makes ensuring that only approved users or processes have access to particular resources or may carry out particular tasks. Access management techniques include:

- User accounts and privileges: Operating systems permit the usage of user accounts to categorise people and grant them with varying levels of access privileges (such as administrator or normal user) to the system's resources.

- Read, write, and execute access to files and directories is controlled by permission settings in operating systems. The owner, group, and other users are often given permissions, giving them fine-grained control over resource access.

- Access control lists (ACLs): ACLs provide for more granular control over resource access by defining access permissions for certain users or groups. They are supported by several operating systems.

2. Authentication:

Prior to allowing access to system resources, authentication confirms the identity of users or processes. It makes sure users are who they say they are and guards against unauthorised access. In operating systems, common authentication techniques include:

- Password authentication: Users enter a password, which is subsequently checked against a hash value that has been saved. The user is given access if the provided password matches the hash that was previously saved.

- Multi-factor authentication (MFA): With this technique, users must supply several pieces of identification information, such as a password and a one-time verification code given to their mobile device, to confirm their identity.

- Biometric authentication: To increase security and user convenience, operating systems are increasingly supporting biometric authentication techniques like fingerprint scanning or face recognition..

3. Authorization:

After successful authentication, authorization establishes what activities or actions a user or process is permitted to carry out. It guarantees that users may access and modify resources with the proper privileges and permissions. Several authorization methods are:

- Role-based access control (RBAC): Rather than assigning permits and privileges based on individual users, RBAC does so based on user roles. Roles are given to users, and those roles have related permissions. This strategy makes administration easier and offers a more controllable authorization structure.

- Mandatory access control (MAC): MAC classifies people and resources according to their sensitivity. Only authorised entities can access resources with matching or acceptable security levels since access choices are dependent on both the user's and the resource's security levels..

- Users can manage access to their resources using discretionary access control (DAC). The owner of the resource has control over who has access to and can alter it.

- In multi-user and multi-process settings, efficient implementation of these security services requires a variety of strategies, including:

Process isolation: Operating systems employ process isolation to prevent unauthorized interactions between processes and ensure that one process cannot interfere with or access another process's resources.

- Memory protection: To separate and safeguard each process's memory area and prevent unauthorised access or alterations, operating systems utilise memory protection methods like virtual memory and address space layout randomization (ASLR).

- Secure inter-process communication (IPC): Operating systems offer secure IPC technologies, such as secure pipes or encrypted communication channels, when processes need to communicate in order to preserve data confidentiality and integrity.

Operating systems may provide a secure environment, safeguard data integrity, and make sure that users and processes have appropriate access to system resources by integrating and effectively applying various security features.

## Different Operating systems

The methods used by various operating systems (OS) for managing files, memories, and processes differ. Let's contrast the security and multi-processing features offered by Linux/Unix, Windows, Android, and macOS:

1. File Management:
    - Linux/Unix: These operating systems include a hierarchical file system structure, with directories represented as files. They use the forward slash (/) to separate directories. The

user, group, and others (ugo) paradigm is used to regulate permissions, and each file has read, write, and execute rights for each category.

- Windows: Windows utilises an individual file system for each disc, such as NTFS or FAT. Backslashes () are used by Windows as a directory separator. It makes use of the Access Control List (ACL) concept, enabling users and groups to have fine-grained control over file permissions.

2. Memory Management:
   - Linux/Unix: Demand-paged virtual memory management is the method used by Linux/Unix systems. They use efficient memory management strategies including paging and swapping. Linux employs a system for managing memory globally.

   - Windows: Demand-paged virtual memory management is also used by Windows. It uses segmentation and paging techniques, with the paging file (page-file.sys) serving as an extension of virtual memory. Local memory management is employed by Windows.

3. Process Management:
   - Linux/Unix: Systems running on Linux/Unix use a process model in which processes are created by utilising the fork() system function. They are in a parent-child relationship, enabling inter-process communication (IPC) techniques like pipes or sockets to be used for process communication. Signals can be used to manage and watch over processes.

   - Windows utilizes a distinct process paradigm. The Create Process() method is used to generate processes. A different Process Identifier (PID) is used for every process. Through tools like named pipes, message queues, and Remote Procedure Calls (RPC), Windows enables process communication. Windows Task Manager offers tools for managing and watching processes.

4. Multi-Process and Security Services:

   - Linux/Unix: These systems offer a strong multi-process architecture that enables the parallel operation of several processes. They provide a range of security services, including access restrictions, user/group management, and file permissions. Systems based on Linux/Unix are renowned for their strong emphasis on security and for providing a wide range of tools for system monitoring and protection.

   - Windows: Windows offers features like process separation, resource allocation, and inter-process communication and supports multi-process architecture. User account control, Windows Defender antivirus, Windows Firewall, and BitLocker disc encryption are just a few of the many security features that Windows provides.

5. Additional Platforms:
   - Android: Android, based on the Linux kernel, has a modified version of the Linux file system, with additional layers for application-specific data storage. It uses the YAFFS2 file system for NAND flash memory. Android implements process management through a

combination of the Linux process model and its own framework for managing activities, services, and application lifecycle. Security services include application sandboxing, permissions model, and app signing.

- macOS: MacOS employs a hierarchical structure akin to Unix and uses the HFS+ or APFS file system. It features a multi-process design and manages processes using a blend of Unix-like processes and the launchd daemon that is exclusive to macOS. Mac OS has a number of security features, including FileVault disc encryption, XProtect malware detection, and Gatekeeper application verification..

## Advantages

1. Access Control: An operating system's procedures for controlling user access to resources are referred to as access control. Here is a comparison of the access control services offered by Windows, macOS, and Unix:

   Windows: Benefits:

Windows: Advantages:

- Robust and granular access control through Access Control Lists (ACLs).
- Integration with Active Directory allows for centralized access control management.
- User-friendly interface for managing access permissions.

Disadvantages:

- Complex configuration and administration process.
- Vulnerable to security breaches if ACLs are misconfigured or not regularly updated.

macOS: Advantages:

- Simple and intuitive access control through POSIX permissions.
- Supports Access Control Lists (ACLs) for more fine-grained access control.
- Secure default settings that limit access to critical system files.

Disadvantages:

- Limited centralized access control management options.
- Less flexibility and granularity compared to Windows or Unix.

Unix: Advantages:

- Strong access control through the file permission system.
- Support for Access Control Lists (ACLs) and extended attributes.
- Provides flexible access control configurations through the command line.

Disadvantages:

- Steeper learning curve for configuring and managing access control.
- May require additional tools or third-party solutions for centralized access control management.

2. Authentication: Authentication is the process of verifying the identity of a user or entity. Let's explore how each platform offers authentication services:

Windows: Advantages:

- Integrated authentication services through Active Directory.
- Support for various authentication mechanisms (e.g., passwords, smart cards, biometrics).
- Seamless integration with enterprise environments.

Disadvantages:

- Higher susceptibility to password-related vulnerabilities (e.g., brute-force attacks).
- Reliance on Active Directory may present challenges for non-Windows environments.

macOS: Advantages:

- Robust authentication using macOS Keychain.
- Support for multiple authentication methods (e.g., passwords, Touch ID).
- Seamless integration with other Apple devices and services.

Disadvantages:

- Limited authentication options for enterprise-level environments.

- Vulnerable to password-based attacks if weak password policies are employed.

Unix: Advantages:

- Flexible authentication mechanisms (e.g., passwords, SSH keys, PAM modules).
- Support for multifactor authentication.
- Extensive customization options for authentication configurations.

Disadvantages:

- Less user-friendly authentication management compared to Windows or macOS.
- May require additional setup and configuration for centralized authentication management.

3. Authorization: Authorization determines the permissions and privileges granted to authenticated users. Let's examine how each platform handles authorization:

Windows: Advantages:

- Fine-grained authorization through user roles, groups, and permissions.
- Integration with Active Directory simplifies authorization management.
- Supports dynamic access control for more advanced authorization scenarios.

Disadvantages:

- Complex authorization configurations.
- Potential for misconfigurations leading to security vulnerabilities.

macOS: Advantages:

- Straightforward authorization through POSIX permissions.
- Support for Access Control Lists (ACLs) for more granular authorization.
- Secure default settings that limit unauthorized access.

Disadvantages:

- Limited centralized authorization management options.
- Less flexibility and scalability compared to Windows or Unix.

Unix: Advantages:

- Highly customizable authorization through file permissions and ownership.
- Support for Access Control Lists (ACLs) and extended attributes.
- Extensive command-line tools for managing authorization.

Disadvantages:

- Requires expertise and careful configuration to prevent authorization errors.
- Lack of centralized management tools can be challenging for large-scale environments.

Summary Comparison Table:

| Platform | Access Control | Authentication | Authorization |
|---|---|---|---|
| Windows | Robust and granular control through ACLs and Active Directory. | Integrated authentication services with various mechanisms. | Fine-grained authorization with user roles, groups, and permissions. |
| macOS | Simple and intuitive access control with POSIX permissions. | Strong authentication through macOS Keychain. | Straightforward authorization with support for ACLs. |
| Unix | Strong access control through file permission system. | Flexible authentication mechanisms with extensive customization. | Highly customizable authorization with support for ACLs and ownership. |

Conclusion: In terms of access control, authentication, and authorization services, each platform—Windows, macOS, and Unix—offers unique benefits and drawbacks. Windows offers strong access control and a wide range of authentication methods, but macOS places a strong emphasis on security and simplicity. Strong access control and customization features are provided by Unix. The requirements, complexity, and environment of the system or organization ultimately determine the platform to be used.