ASSIGNMENT 1

NAME: RESMI SAJI

SUBMITTTED TO: MISS HARITHA

AIM:

To familiarize and understand the use and functioning of System Calls used for Operating system in Windows and Unix.

SYSTEM CALLS:

System calls are used for the interaction between an application and the operating system, where they act as a primary method for user level program to request services from operating system's kernel.

It acts as an entry point into the operating system, that involves operations that require privileged access to hardware or system resources, which are not directly accessible by user-space programs due to security and stability reasons.

System calls are implemented at the kernel level and exposed to user-space applications through an Application Programming Interface (API). When an application invokes a system call, it triggers a context switch from user mode to kernel mode, where the operating system has full access to hardware resources

DIFFERENT TYPES OF SYETEM CALLS IN WINDOWS AND LINUX:

PROCESS	WINDOWS	UNIX
Process control	• Createprocess() • ExitProcess() • WaitForSingleObject()	• Fork() • Exit() • Wait()
File Manipulation	• CreateFile() • ReadFile() • WriteFile() • CloseHandle()	• Open() • Read() • Write() • Close()
Device Management	• SetConsoleMode() • ReadConsole() • WriteConsole()	• Ioctl() • Read() • Write()
Information Maintenance	• GetCurrentProcessID() • SetTimer() • Sleep()	• Getpid() • Alarm() • Sleep()
Communication	• CreatePipe() • CreateFileMapping() • MapViewOfFile()	• Pipe() • Shmget() • Mmap()
Protection	• SetFileSecurity() • InitializseSecurityDescriptor() • SetSecurityDescriptorGroup()	• Chmod() • Umask() • Chown()

VARIOUS SYSTEM CALLS:

1. **Process Control:**

system call that is used to direct the processes that includes process creation, process termination, process allocation and deallocation etc.

2. **File Manipulation:**

File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.

3. **Device Management:**

Process of implementation, operation, and maintenance of a device by an operation system.

4. **Information Maintenance:**

It manages and preserve data within computer systems. This is a special set of system calls used to manage information and perform various tasks related to system information, time and date, and system configuration.

5. **Communication:**

Used for inter process communications which create, delete communications connections, send, receive message helps OS to transfer status information, attach or detach remote devices.

6. **Protection:**

Protection refers to a mechanism which controls the access of programs, processes, or users to **the** resources defined by a computer system.

SYSTEM CALLS FOR WINDOWS:

1. **Process Control:**
   - **CreateProcess()** : The CreateProcess() function is a fundamental API in Windows programming used to create a new process and its primary thread. It is part of the Windows API and provides a way to start a new executable file.
   - **ExitProcess() :** The ExitProcess() function is used to terminate the current process and its threads. It is part of the Windows API and allows a process to end its execution, optionally returning an exit code to the operating system.
   - **WaitForSingleObject()** : The WaitForSingleObject () function is a Windows API function that is used to wait for a specified object to be signalled. It can be used to wait for a variety of objects, including process and thread handles, event objects, mutexes, and semaphores.
2. **File Manipulation**
   - **CreateFile()** : The CreateFile() function is used to create or open a file or I/O device, such as a file, directory, or console.
   - **ReadFile() :** The ReadFile() function reads data from a file or I/O device into a buffer.
   - **WriteFile()** : The WriteFile() function writes data to a file or I/O device from a buffer, performing write operations in Windows applications, allowing you to write data to files, pipes, or other types of I/O devices.

- **CloseHandle() :** The CloseHandle() function closes an open object handle, such as a file handle. This function is essential for resource management in Windows applications, as it releases the resources associated with the handle and ensures that the system resources are properly cleaned up.

3. **Device Management**
   - **SetConsoleMode() :** The SetConsoleMode() function is used to set the input mode of a console's input buffer. This function affects how the console processes input, such as keyboard input, and determines how input is read and interpreted by the console.

   - **ReadConsole()** : The ReadConsole() function is used to read data from the console's input buffer. It allows you to directly read user input from the console, bypassing higher-level input functions. This can be particularly useful for applications that need to process input in a specialized way or **handle raw input.**

   - **WriteConsole() : The WriteConsole() function is used to write a character string to the console's output buffer. This function is useful for applications that need to output text to the console or manipulate the console's display directly.**

4. **Information Maintenance**
   - **GetCurrentProcessID() :** The GetCurrentProcessId() function retrieves the process identifier (PID) of the calling process. This PID is a unique value assigned by the operating system to identify the process.

   - **SetTimer()** : The SetTimer() function is used to create a timer that triggers at specified intervals, sending a message or calling a callback function when the timer elapses. This function is often used for periodic tasks, animations, or time-based operations within a Windows application.

   - **Sleep() :** The Sleep() function is used to suspend the execution of the current thread for a specified number of milliseconds. This can be useful for various purposes, like delaying operations, throttling execution, or creating time delays in a program

5. **Communication**
   - **CreatePipe() :** The CreatePipe() function creates a pipe, which is a unidirectional communication channel that can be used for inter-process communication (IPC). The pipe allows data to flow from one process to another or between threads within the same process.

   - **CreateFileMapping() :** The CreateFileMapping() function create a named or unnamed file mapping object, which enables processes to share data by mapping a file or a section of memory into the address space of the process. This function is particularly useful for inter-process communication (IPC) or for accessing files in a memory-mapped fashion.

   - **MapViewOfFile() :** The MapViewOfFile() function is used to map a view of a file mapping object into the address space of a process. This allows the process to access the contents of the file mapping as if it were part of the process's memory, enabling efficient file I/O operations and inter-process communication (IPC).

6. **Protection**
   - **SetFileSecurity() :** The SetFileSecurity() function is used to set the security attributes of a file or directory. This includes specifying access control lists (ACLs) that define the permissions for

users and groups on the file or directory. The function allows you to modify the security descriptor associated with a file or directory, which controls access and audit permissions.

- **InitializseSecurityDescriptor()** : The InitializeSecurityDescriptor() function is used to initialize a SECURITY_DESCRIPTOR structure. This function sets up the security descriptor to a known state so that it can be used to set or modify security attributes for files, directories, or other securable objects.
- **SetSecurityDescriptorGroup() :** The SetSecurityDescriptorGroup() function is used to set the group information in a SECURITY_DESCRIPTOR structure. This function allows you to specify the group for which access control permissions are applied within the security descriptor.

SYSTEM CALLS FOR UNIX:

1. **Process Control**
   - **Fork() :** The fork() function is a system call used to create a new process. The new process, referred to as the child process, is a duplicate of the calling process, known as the parent process. Both processes will continue executing from the point where the fork call was made, but they have separate process identifiers (PIDs) and memory spaces.

   - **Exit() :** The exit() function performs cleanup operations, such as flushing and closing file streams, releasing resources, and then terminates the process with a specified exit status. This function provides a way to end a program and return an exit status to the operating system.

   - **Wait() :** The wait() function is used by a process to wait for state changes in a child process.It is used to wait for a child process to terminate and to retrieve its exit status. This function process synchronization and management, allowing a parent process to obtain information about the termination of its child processes.

2. **File Manipulation**
   - **Open() :** The open() function is used to open a file or device and obtain a file descriptor, performs subsequent operations on the file, such as reading, writing, or closing. It is a low-level system call that provides a way to interact with files and devices at a more fundamental level than standard I/O functions.

   - **Read() :** The read() function is used to read data from a file descriptor into a buffer, performing I/O operations on files, pipes, and devices. The read function reads a specified number of bytes from the file associated with the file descriptor and stores them in a buffer.

   - **Write() :** The write() function is used to write data to a file descriptor, allows to send data to a file, pipe, or device. The function writes a specified number of bytes from a buffer to the file descriptor.

   - **Close() :** The close() function is used to close a file descriptor, releasing the resources associated with it. Closing a file descriptor is important for proper resource management and ensuring that system resources are freed when they are no longer needed.

3. **Device** Management
   - **Ioctl() :** The ioctl() function is used to control or configure device characteristics or perform various I/O operations that are not covered by standard system calls like read and write. It perform device-specific operations by passing an integer command code and optional parameters.

- **Read() :** The read() function is used to read data from a file descriptor into a buffer, allows to retrieve data from files, pipes, sockets, and devices.

- **Write() :** The write() function is used to write data to a file descriptor. It is a fundamental system call for performing output operations on files, pipes, sockets, and devices.

4. **Information Maintenance**
   - **Getpid() :** The getpid() function is used to obtain the process ID (PID) of the calling process. The process ID is a unique identifier assigned by the operating system to each process.

   - **Alarm() :** The alarm() function is used to schedule an asynchronous signal to be sent to the calling process after a specified number of seconds. This function is useful for setting timers and implementing timeouts**.**

   - **Sleep() :** The sleep() function is used to pause the execution of the calling thread for a specified number of seconds. It is a simple way to delay execution and can be useful for implementing timeouts.

5. **Communication**
   - **Pipe():** The pipe() function creates a unidirectional communication channel between processes. It provides a way for one process to send data to another process using a pipe.

   - **Shmget() :** The shmget() function is used to allocate a shared memory segment. Shared memory is a form of inter-process communication (IPC) that allows multiple processes to access the same memory space, making it useful for exchanging data efficiently between processes.

   - **Mmap() :** The mmap() function is used to map files or devices into memory, allowing processes to access the contents of a file or device as if it were a part of memory. This can facilitate efficient file I/O and inter-process communication**.**

6. **Protection**
   - **Chmod() :** The chmod() function changes the permissions of a file or directory. It modifies the access control of the specified file or directory based on the mode provided.

   - **Umask()** : The umask() function sets or retrieves the process's umask value. The umask (user file creation mask) is a bitmask that determines the default permissions that are not set when creating new files or directories. It essentially controls the default permissions by masking out specific permission bits.

   - **Chown() : The chown() function changes the ownership of a file or directory. It allows a process to modify the user and/or group ownership of a specified file or directory.**