The `close()` system call in Linux is used to deallocate a file descriptor, making it available for reuse by subsequent `open()` or similar functions that allocate file descriptors. It serves to release system resources associated with an open file, which were previously acquired when the file was opened using `open()`.

Syntax:

C
```c
int close(int fd);
```

Parameters:

- `fd`: The file descriptor of the file that is to be closed.

Return Value:

- `0` on success.
- `-1` on error, with `errno` set to indicate the specific error.

Functionality:

When `close(fd)` is called, the operating system performs the following actions:

- **Decrements Reference Count:**

  The kernel maintains a reference count for each file table entry. `close()` decrements this count.

- **Resource Deallocation:**

  If the reference count for the file table entry reaches zero (meaning no other process or part of the current process is referencing this file), the kernel deallocates the associated resources, including the file table entry and any related kernel buffers.

- **File Descriptor Table Update:**
  The entry in the process's file descriptor table corresponding to `fd` is marked as free, making that file descriptor number available for new `open()` calls.

Importance:

It is crucial to call `close()` for every file descriptor obtained through `open()` or similar calls to prevent resource leaks. Failing to close file descriptors can lead to exhaustion of available file descriptors and other system resources, potentially impacting system stability and performance. Even if a process terminates, open file descriptors are automatically closed by the operating system; however, explicitly closing them during the program's execution is considered good practice for resource management.