**Assignment – 4**  
**Aim:** To study and learn about various system calls  
**To perform:** Comprehensive study of different categories of Linux system calls, categorized as  
  
Linux system calls are essential interfaces between user applications and the kernel, enabling processes to request services like process management, file operations, and device control. Below is a detailed study of key system calls across five categories, including examples and usage scenarios.

**Process Management System Calls**

These calls handle process creation, execution, and termination.

**fork()**  
Creates a new process by duplicating the current process. The child process starts execution from the return point of fork(), with the parent receiving the child’s PID and the child receiving 0[1](https://percona.community/blog/2021/01/04/fork-exec-wait-and-exit/).

**exec()**  
Replaces the current process image with a new program. For example, execvp("ls", args) runs the ls command in the current process[1](https://percona.community/blog/2021/01/04/fork-exec-wait-and-exit/).

**wait()**  
Suspends the parent process until a child terminates. It retrieves the child’s exit status using waitpid()[1](https://percona.community/blog/2021/01/04/fork-exec-wait-and-exit/).

**exit()**  
Terminates a process and returns an exit status to the parent. All resources (e.g., open files) are released[1](https://percona.community/blog/2021/01/04/fork-exec-wait-and-exit/).

**Example Code**

c

#**include** <unistd.h>

#**include** <sys/wait.h>

**int** main() {

pid\_t pid = fork();

**if** (pid == 0) { *// Child process*

execlp("ls", "ls", NULL);

exit(1); *// Only reached if exec fails*

} **else** { *// Parent process*

wait(NULL); *// Wait for child to finish*

}

**return** 0;

}

**File Management System Calls**

These manage file operations like opening, reading, and writing.

**open()**  
Opens a file and returns a file descriptor. Modes include O\_RDONLY (read-only) and O\_WRONLY (write-only)[3](https://www.educative.io/answers/introduction-to-system-calls).

**read()/write()**  
Transfer data between a file descriptor and a buffer.

**close()**  
Releases a file descriptor and updates metadata[3](https://www.educative.io/answers/introduction-to-system-calls).

**Example Code**

c

#**include** <fcntl.h>

**int** main() {

**int** fd = open("file.txt", O\_WRONLY | O\_CREAT, 0644);

write(fd, "Hello", 5);

close(fd);

**return** 0;

}

*This code writes "Hello" to file.txt*[*3*](https://www.educative.io/answers/introduction-to-system-calls)*.*

**Device Management System Calls**

These interact with hardware devices.

**ioctl()**  
Performs device-specific operations (e.g., setting serial port baud rates). For example, configuring a modem’s transmission speed[5](https://ufal.mff.cuni.cz/~jernej/2018/docs/predavanja06.pdf).

**read()/write()**  
Reused from file management to interact with device files (e.g., /dev/ttyS0).

Network Management System Calls

These handle socket communication.

**socket()**  
Creates a communication endpoint (e.g., AF\_INET for IPv4).

**connect()**  
Initiates a connection to a remote socket.

**send()/recv()**  
Transmit or receive data over a network socket.

**System Information Management System Calls**

Retrieve system and process details.

**getpid()**  
Returns the current process ID.

**getuid()**  
Returns the user ID of the process owner.

**gethostname()**  
Fetches the system’s hostname.

**sysinfo()**  
Provides system statistics (e.g., memory usage).

This study covers foundational Linux system calls across key categories, illustrating their roles in process control, file handling, device interaction, networking, and system monitoring. Mastery of these calls is critical for developing efficient system-level applications.

Linux system calls are essential interfaces between user applications and the kernel, enabling processes to request services like process management, file operations, and device control. Below is a detailed study of key system calls across five categories, including examples and usage scenarios.

## Process Management System Calls

These calls handle process creation, execution, and termination.

**fork()**  
Creates a new process by duplicating the current process. The child process starts execution from the return point of fork(), with the parent receiving the child’s PID and the child receiving 0[1](https://percona.community/blog/2021/01/04/fork-exec-wait-and-exit/).

**exec()**  
Replaces the current process image with a new program. For example, execvp("ls", args) runs the ls command in the current process[1](https://percona.community/blog/2021/01/04/fork-exec-wait-and-exit/).

**wait()**  
Suspends the parent process until a child terminates. It retrieves the child’s exit status using waitpid()[1](https://percona.community/blog/2021/01/04/fork-exec-wait-and-exit/).

**exit()**  
Terminates a process and returns an exit status to the parent. All resources (e.g., open files) are released[1](https://percona.community/blog/2021/01/04/fork-exec-wait-and-exit/).

**Example Code**

c

#**include** <unistd.h>

#**include** <sys/wait.h>

**int** main() {

pid\_t pid = fork();

**if** (pid == 0) { *// Child process*

execlp("ls", "ls", NULL);

exit(1); *// Only reached if exec fails*

} **else** { *// Parent process*

wait(NULL); *// Wait for child to finish*

}

**return** 0;

}

## File Management System Calls

These manage file operations like opening, reading, and writing.

**open()**  
Opens a file and returns a file descriptor. Modes include O\_RDONLY (read-only) and O\_WRONLY (write-only)[3](https://www.educative.io/answers/introduction-to-system-calls).

**read()/write()**  
Transfer data between a file descriptor and a buffer.

**close()**  
Releases a file descriptor and updates metadata[3](https://www.educative.io/answers/introduction-to-system-calls).

**Example Code**

c

#**include** <fcntl.h>

**int** main() {

**int** fd = open("file.txt", O\_WRONLY | O\_CREAT, 0644);

write(fd, "Hello", 5);

close(fd);

**return** 0;

}

This code writes "Hello" to *file.txt*[*3*](https://www.educative.io/answers/introduction-to-system-calls)*.*

## 

## Device Management System Calls

These interact with hardware devices.

**ioctl()**  
Performs device-specific operations (e.g., setting serial port baud rates). For example, configuring a modem’s transmission speed[5](https://ufal.mff.cuni.cz/~jernej/2018/docs/predavanja06.pdf).

**read()/write()**  
Reused from file management to interact with device files (e.g., /dev/ttyS0).

## Network Management System Calls

These handle socket communication.

**socket()**  
Creates a communication endpoint (e.g., AF\_INET for IPv4).

**connect()**  
Initiates a connection to a remote socket.

**send()/recv()**  
Transmit or receive data over a network socket.

## System Information Management System Calls

Retrieve system and process details.

**getpid()**  
Returns the current process ID.

**getuid()**  
Returns the user ID of the process owner.

**gethostname()**  
Fetches the system’s hostname.

**sysinfo()**  
Provides system statistics (e.g., memory usage).

This study covers foundational Linux system calls across key categories, illustrating their roles in process control, file handling, device interaction, networking, and system monitoring. Mastery of these calls is critical for developing efficient system-level applications.