**SYSTEM CALLS**

**What are System Calls?**

A **System Call** is a way for a **program (user mode)** to request a service from the **Operating System’s kernel (kernel mode)**.

Since user programs cannot directly access hardware or sensitive resources (for protection), they must use system calls as an **interface** between the user and the OS.

**Types of System Calls**

System calls are usually grouped into 5 categories:

1. **Process Control**

Create, execute, terminate processes.

Example: fork(), exec(), exit(), wait().

Example in Linux: fork() creates a new process.

1. **File Management**

Operations on files (read, write, open, close).

Example: open(), read(), write(), close(), delete().

Example: open("file.txt", O\_RDONLY).

1. **Device Management**

Request and release devices, read/write device files.

Example: ioctl(), read(), write().

1. **Information Maintenance**

Get or set system information, attributes.

Example: getpid() (get process ID), alarm(), time().

1. **Communication**

Inter-process communication (IPC).

Example: pipe(), shmget(), mmap(), msgsnd(), recv().

**How System Calls Work**

A program issues a **system call** (e.g., read a file).

The **CPU switches from user mode to kernel mode** (via a trap/interrupt).

The **OS executes the requested service**.

The **result is returned** to the user program.

CPU switches back to **user mode**.

1. **fork() – Create a Child Process**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

perror("fork failed");

exit(1);

}

else if (pid == 0) {

printf("This is the Child Process. PID = %d, PPID = %d\n", getpid(), getppid());

}

else {

printf("This is the Parent Process. PID = %d, Child PID = %d\n", getpid(), pid);

}

return 0;

}

**OUTPUT:**

This is the Parent Process. PID = 632, Child PID = 633

This is the Child Process. PID = 633, PPID = 632

1. **exit() – Terminate a Process with Status**

**#include <stdio.h>**

**#include <unistd.h>**

**#include <stdlib.h>**

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

**int main() {**

**pid\_t pid = fork();**

**if (pid == 0) {**

**printf("Child process exiting with code 5\n");**

**exit(5); // child exits with status 5**

**} else {**

**int status;**

**wait(&status);**

**if (WIFEXITED(status)) {**

**printf("Parent received child's exit status: %d\n", WEXITSTATUS(status));**

**}**

**}**

**return 0;**

**}**

****OUTPUT:****

**Child process exiting with code 5**

**Parent received child's exit status: 5**

1. **getpid() – Get Process ID**

**#include <stdio.h>**

**#include <unistd.h>**

**int main() {**

**printf("Current Process PID = %d\n", getpid());**

**printf("Parent Process PID = %d\n", getppid());**

**return 0;**

**}**

**OUTPUT:**

**Current Process PID = 2411**

**Parent Process PID = 2404**

1. **wait() – Parent Waits for Child**

**#include <stdio.h>**

**#include <unistd.h>**

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

**#include <stdlib.h>**

**int main() {**

**pid\_t pid = fork();**

**if (pid == 0) {**

**printf("Child process (PID = %d) running...\n", getpid());**

**sleep(2);**

**printf("Child process finished.\n");**

**exit(0);**

**} else {**

**printf("Parent waiting for child...\n");**

**wait(NULL);**

**printf("Parent resumes after child finished.\n");**

**}**

**return 0;**

**}**

****OUTPUT:****

**Child process (PID = 2669) running...**

**Parent waiting for child...**

**Child process finished.**

**Parent resumes after child finished.**

1. **close() – Close a File Descriptor**

**#include <stdio.h>**

**#include <unistd.h>**

**#include <fcntl.h>**

**int main() {**

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

**if (fd < 0) {**

**perror("open failed");**

**return 1;**

**}**

**write(fd, "Hello, System Calls!\n", 22);**

**if (close(fd) == 0) {**

**printf("File closed successfully.\n");**

**} else {**

**perror("close failed");**

**}**

**return 0;**

**}**