**Example of fork():**

#include <stdio.h>

#include <unistd.h>

int main() {

pid\_t pid = fork();

if (pid > 0) {

printf("Parent Process: PID = %d, Child PID = %d\n", getpid(), pid);

} else if (pid == 0) {

printf("Child Process: PID = %d\n", getpid());

} else {

printf("Fork failed!\n");

}

return 0;

}

**3. The exec() System Call**

While fork() creates a new process, it still runs the same program as the parent. To replace the current process image with a new one, we use exec(). The exec() family of functions loads a new program into the current process space and starts execution.

**Common exec() Variants:**

* execl(), execv(), execle(), execve()
* They differ in how they pass arguments and environment variables.

**Example of exec():**

#include <stdio.h>

#include <unistd.h>

int main() {

printf("Executing ls command\n");

execl("/bin/ls", "ls", "-l", NULL);

printf("This line will not execute if execl succeeds\n");

return 0;

}

**4. Combining fork() and exec()**

A typical pattern in Linux involves using fork() to create a new process and exec() to replace its image with a new program.

**Example:**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid = fork();

if (pid == 0) {

// Child process

execl("/bin/ls", "ls", "-l", NULL);

} else {

// Parent process

wait(NULL);

printf("Child process completed.\n");

}

return 0;

}

**5. Conclusion**

Understanding fork() and exec() is essential for working with Linux process creation. fork() duplicates a process, while exec() replaces a process image with a new program. These concepts are widely used in system programming, multiprocessing, and creating efficient applications that leverage process control in Linux.