Nepal College of Information Technology
Balkumari, Lalitpur
(Affiliated to Pokhara University)

A Lab Report
On

Understanding Process Creation and Execution in UNIX: Fork and Exec System Calls

Submitted as partial fulfillment of requirement of the curriculum of Bachelor's of Engineering in Software Engineering (6th Semester)

Submitted by:
Harsh Chaudhary Kalwar
(221715)

Submitted to:
Madan Bhandari

Date:
16th June, 2025

Objective:

The objective of this lab was to explore how UNIX-based systems handle process creation and execution using the fork(), exec(), and wait() system calls. Through practical implementation, we observed how parent and child processes are managed and how control is transferred using execlp().

Lab Tasks and Execution:

1. Exploring fork() and wait() with Message Printing:

File: fork.c

Description:

· Created a child process using fork().

· Parent process waited for the child to complete using wait().

· Both parent and child printed messages to track process flow.

Key Code Snippet:

```
pid_t pid = fork();
if (pid > 0){
wait(NULL);
printf("Hello from parent %5d\n", getpid());
}
else if (pid == 0){
printf("Hello from child %5d and my parent id %5d\n", getpid(),
getppid());
}
```

Observation:

· On execution, both child and parent process printed messages.

· The parent printed only after the child completed, ensuring sequential output due to wait().

2. Simple Fork Test with Basic Validation:

File: fork1.c

Description:

· Similar to the first file but added error checking with if(pid < 0).

· Printed unique messages from both child and parent after a successful fork().

Key Code Snippet:

```
if(pid < 0){
printf("Failed");
}
if (pid == 0){
printf("Hello from child %d\n", getpid());
}
else{
wait(NULL);
printf("Hello from parent %d\n", getpid());
}
```

Observation:

· Confirmed that fork() returns 0 in the child and child PID in the parent.
· Ensured that parent process waited until the child finished.

3. Executing a New Program from Child Using execlp():

File: childp.c
Description:
· After a successful fork(), the child process used execlp() to execute the ls -al command.
· The parent waited and printed a message after the child completed.

Key Code Snippet:

```
if (pid == 0) {
execlp("/bin/ls", "ls", "-al", NULL);
}
else {
wait(NULL);
printf("Child Complete %d\n", getpid());
}
```

Observation:

· Child process replaced its execution image with the ls -al command output.
· Parent printed confirmation after child execution finished.
· Demonstrated how exec can run a completely different program in a forked process.

Output / Observations:

- fork() creates a new child process with its own PID.
- wait() in the parent process delays its execution until the child finishes.
- execlp() successfully replaced the child's process image to run ls -al.
- getpid() and getppid() were used effectively to observe process hierarchy.

Conclusion:

This lab deepened our understanding of how processes are handled in a UNIX/Linux system. We observed how fork() duplicates a process, wait() synchronizes execution, and exec() allows a child process to launch another program. This foundation is critical for systems programming and multithreaded or multiprocess applications.