Operating Systems

Lab Manual # 4 (Extented)

Process Creation using `fork` and `exec`

Objectives:

- Understand the concepts of process creation and process hierarchy in Unix-like operating systems.
- Learn how to use the `fork` system call to create a new process.
- Explore the role of the `exec` system call in replacing the current process image.
- Observe the interaction between parent and child processes.

Pre-requisites:

- Basic understanding of GCC and C language in Ubuntu.
- Completion of Lab 4.
- Ubuntu operating system (either installed on your computer or in a virtual machine).

Lab Tasks:

Exercise 1: Understanding Process Creation

- To create a C program that demonstrates the basic concepts of process creation using the `fork` system call.

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t childPid = fork();

    if (childPid == 0) {
        // Child process
        printf("This is Child Process: PID = %d\n", getpid());

        // Execute a different program (ls in this case)
        execl("/bin/ls", "ls", "-l", (char *)NULL);
```

```
        // If execl fails
        perror("Error in exec");
    } else if (childPid > 0) {
        // Parent process
        printf("This is Parent Process: PID = %d\nWaiting for Child Process to complete!\n", getpid());
        wait(NULL);
        printf("The Child process completed.\n");
    } else {
        // Fork failed
        perror("Error in fork");
        return 1;
    }

    return 0;
}
```

Exercise 2: Instructions to Run

- Open a terminal and navigate to the directory where you saved the program.
- Compile the program using the following command:

```
$ gcc process_creation.c -o process_creation
```

- Execute the program:

```
$ ./process_creation
```

Explanation:

1. **Understanding `fork`:**
   - The `fork` system call creates a new process by duplicating the existing process. The child process is an identical copy of the parent process.
2. **Child Process Execution (`execl`):**
   - In the child process, the `execl` system call replaces the current process image with a new one. In this example, it executes the `ls -l` command.
3. **Parent Process Waiting (`wait`):**
   - The parent process uses the `wait` system call to wait for the child process to complete before continuing its execution.

Sample Output:

```
This is Parent Process: PID = 1234
Waiting for Child Process to complete!
This is Child Process: PID = 1235
<output of ls -l command>
The Child process completed.
```

Exercise 3: Modify and Experiment

- Modify the program to execute a different command in the child process (e.g., `date`).
- Re-compile the program and observe the changes in the output.
- Experiment with different commands in the child process and analyze the results.

Exercise 4: Instructions to Run

- Open the program in a text editor and make the necessary modifications.
- Save the changes and re-compile the program.
- Execute the modified program:

```
$ ./process_creation
```

Explanations:

- By modifying the program and changing the command executed in the child process, you can observe how the child process behavior affects the overall output.

Exercise 5: Handling Errors

- To understand error handling in process creation.
- Intentionally introduce an error in the `execl` system call in the child process (e.g., provide an incorrect path).
- Observe how the program handles the error.

Exercise 6: Instructions

- Open the program in a text editor and intentionally introduce an error in the `execl` system call.
- Save the changes and re-compile the program.
- Execute the modified program
- Observe the error message and understand how the program responds to the error.

Explanations:

- Introducing an error in the `execl` system call simulates a scenario where the child process cannot execute the specified command.

Sample Output:

```
This is Parent Process: PID = 1234
Waiting for Child Process to complete!
This is Child Process: PID = 1235
Error in exec: No such file or directory
```

Conclusions:

Congratulations! You've successfully completed this lab, in this lab exercise, you gained hands-on experience with process creation using the `fork` system call and executing a different program in the child process using `exec`. Experimenting with different commands and error scenarios enhances your understanding of process management in Unix-like operating systems.