LAB 3 : SYSTEM CALLS

Nirajan Bekoju
PUL076BCT039
076bct039.nirajan@pcampus.edu.np

Nishant Luitel
PUL076BCT041
076bct041.nishant@pcampus.edu.np

Nabin Da Shrestha
PUL076BCT037
076bct037.nabin@pcampus.edu.np

Prakash Chaulagain
PUL076BCT045
076bct045.prakash@pcampus.edu.np

I. Introduction

In an operating system, process creation is the process of creating a new process from an existing process. This is typically done using system calls such as fork() and exec() in the C programming language. Here's a step-by-step description of the process creation process using C:

1) The parent process calls the fork() system call to create a new child process.
2) The fork() system call creates a new process by duplicating the parent process. The child process is an exact copy of the parent process, including all the memory, data, and code.
3) After the fork() call, the parent and child processes continue executing independently. They share the same code and data, but have separate copies of the memory.
4) The child process can modify its own copy of the memory without affecting the parent process or any other processes.
5) The parent process can use the returned process ID from the fork() call to identify the child process.
6) The child process can use the exec() system call to load a new program into its memory space. This replaces the child process's memory and data with the new program and starts executing it.
7) The parent process can wait for the child process to finish using the wait() system call. This suspends the parent process until the child process terminates.
8) When the child process terminates, it returns an exit code to the parent process using the exit() system call. The exit code indicates whether the child process completed successfully or encountered an error.

Overall, the process creation process in an operating system involves creating a new process from an existing one, allowing the new process to execute independently, and managing the communication between the parent and child processes. The C programming language provides system calls such as fork() and exec() to facilitate this process.

II. Process Creation

Aim : To write a program to create a process in UNIX.

Algorithm :

STEP 1: Start the program.
STEP 2: Declare pid as integer.
STEP 3: Create the process using Fork command.
STEP 4: Check pid is less than 0 then print error else if pid is equal to 0 then execute command else parent process wait for child process.
STEP 5: Stop the program.

Program :

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
    pid_t pid, mypid, myppid;
    pid = getpid();

    printf("Before fork: Process id is %d\n", pid);
    pid = fork();
    printf("After fork: Process id is %d\n", pid);
```

```
    if(pid < 0){
        perror("fork() failure \n");
        return 1;
    }

    if(pid == 0){
        printf("This is child process\n");
        mypid = getpid();
        myppid = getppid();
        printf("Process id is %d and PPID is %d\n", mypid, myppid);
    }
    else{
        sleep(2);
        printf("This is parent process\n");
        mypid = getpid();
        myppid = getppid();
        printf("Process id is %d and PPID is %d \n", mypid, myppid);
        printf("Newly created process id or child pid is %d\n", pid);
    }
    return 0;
}
```

Output :

```
Before fork: Process id is 32828
After fork: Process id is 0
This is child process
Process id is 32838 and PPID is 32828
After fork: Process id is 32838
This is parent process
Process id is 32828 and PPID is 32813
Newly created process id or child pid is 32838
```

III. Executing a command

Aim : To write a program for executing a command.

Algorithm :

STEP 1 :Start the program.
STEP 2: Execute the command in the shell program using exec ls.
STEP 3: Stop the execution.

Program :

```
echo "Program for executing UNIX command using shell programming "
echo "Welcome"
ps
```

Output:

```
Program for executing UNIX command using shell programming
Welcome
    PID TTY          TIME CMD
  33095 pts/4    00:00:00 bash
  33110 pts/4    00:00:00 bash
  33111 pts/4    00:00:00 ps
```

IV. Sleep Command

Aim : To create child with sleep command.

Algorithm :

STEP 1: Start the program.
STEP 2: Create process using fork and assign into a variable.

STEP 3: If the value of variable is less than zero : print not create and greater than 0 : process create and else print child create.
STEP 4: Create child with sleep of 2.
STEP 5: Stop the program.

Program :

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h> // for exit(0) command

int main(){
    int pid = fork();
    if (pid < 0){
        printf("Process not created");
        return -1;
    }
    if (pid == 0){
        sleep(2);
        printf("Child process created");
    }
    else{
        printf("Parent Process created");
    }
}
```

Ouput :

```
Parent Process created
Child process created
```

V. SLEEP COMMAND USING GETPID

Aim : To create child with sleep command using getpid.

Algorithm :

STEP 1: Start the execution and create a process using fork() command.
STEP 2: Make the parent process to sleep for 10 seconds.
STEP 3:In the child process print it pid and it corresponding pid.
STEP 4: Make the child process to sleep for 5 seconds.
STEP 5: Again print it pid and it parent pid.
STEP 6: After making the sleep for the parent process for 10 seconds print it pid.
STEP 7: Stop the execution.

Program :

```
#include <stdio.h>
#include <unistd.h>

int main(){
    int ppid;
    int pid = fork();
    if (pid < 0){
        printf("Process not created\n");
        return -1;
    }
    if (pid > 0){
        printf("Parent process\n");
        sleep(10);
        pid = getpid();
        printf("Parent process id is %d\n", pid);
    }
    else{
```

```
        printf("Child process\n");
        pid = getpid();
        ppid = getppid();
        printf("child process id %d and parent process id is %d\n", pid, ppid);
        sleep(5);
        printf("Child process after 5s sleep \n");
        printf("child process id %d and parent process id is %d\n", pid, ppid);
    }
}
```

Output :

```
Parent process
Child process
child process id 33852 and parent process id is 33841
Child process after 5s sleep
child process id 33852 and parent process id is 33841
Parent process id is 33841
```

VI. Signal Handling

Aim : To write a program for signal handling in UNIX.

Algorithm :

STEP 1:start the program
STEP 2:Read the value of pid.
STEP 3:Kill the command surely using kill-9 pid.
STEP 4:Stop the program.

Program :

```
echo "Program for performing KILL operations"
ps
echo "enter the pid"
read pid
kill -9 $pid
echo "finished"
```

Ouput :

```
Program for performing KILL operations
    PID TTY          TIME CMD
  34080 pts/4    00:00:00 bash
  34112 pts/4    00:00:00 bash
  34113 pts/4    00:00:00 ps
enter the pid
34112
Killed
```

VII. Wait Command

Aim : To perform wait command using c program.

Algorithm :

STEP 1: Start the execution
STEP 2: Create process using fork and assign it to a variable
STEP 3: Check for the condition pid is equal to 0
STEP 4: If it is true print the value of i and terminate the child process
STEP 5: If it is not a parent process has to wait until the child terminate
STEP 6: Stop the execution

Program :

```
#include <stdio.h>
#include <unistd.h>
```

```
#include <sys/types.h>
#include <sys/wait.h>

int main(){
    int pid, ppid;
    pid = fork();
    if(pid < 0){
        printf("Process not created");
        return -1;
    }
    if (pid == 0){
        printf("Child Process \n");
        pid = getpid();
        printf("Child process id is %d\n", pid);
        return 0;
    }
    else{
        wait(NULL);
        printf("Parent Process executing\n");
    }
    return 0;
}
```

Ouput :

```
Child Process
Child process id is 34400
Parent Process executing
```

VIII. Conclusion

In conclusion, this lab report focused on the implementation of process creation and management in Linux operating systems. Through the use of system calls and C programming, we were able to create child processes, obtain their process IDs, and manipulate them through the use of various commands such as sleep, wait, and kill. The implementation of the ”ps” command and the creation of a shell program to simulate it further demonstrated the versatility and power of system calls in process management. Overall, this lab provided a valuable opportunity to explore the intricacies of process creation and management in a Linux environment.