Operating System Lab

1.Analyze and execute different types of system calls for process creation and termination.

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

int main() {
    pid_t pid;
    int status;

    // fork() system call
    pid = fork();

    if (pid < 0) {
        printf("Error: fork() failed.\n");
        return 1;
    } else if (pid == 0) {
        // child process

        printf("This is the child process with PID: %d\n", getpid());
        printf("Parent process PID: %d\n", getppid());

        // exec() system call
        execlp("/bin/ls", "ls", NULL);

        printf("This should not be printed if exec() is successful.\n");
        return 0;
    } else {
        // parent process

        printf("This is the parent process with PID: %d\n", getpid());
        printf("Child process PID: %d\n", pid);

        // wait() system call
        wait(&status);

        printf("Child process exited with status: %d\n", status);

        return 0;
    }
}
```

In this program, we use three different system calls for process creation and termination:

fork(): This system call is used to create a new process by duplicating the calling process. The child process and the parent process will have different process IDs (PIDs), and they will run independently of each other.

exec(): This system call is used to replace the current process image with a new process image. In the child process, we use execlp() to replace the current process image with the ls command, which lists the contents of the current directory.

wait(): This system call is used by the parent process to wait for the child process to exit. The status of the child process is stored in the status variable.

When you run this program, you should see output similar to the following:

yaml
Copy code
```
This is the parent process with PID: 1234
Child process PID: 1235
This is the child process with PID: 1235
Parent process PID: 1234
file1.txt file2.txt file3.txt
Child process exited with status: 0
```
This output shows that the parent process creates a child process using the fork() system call, and then waits for the child process to exit using the wait() system call. The child process uses the exec() system call to replace its process image with the ls command, which lists the contents of the current directory.