**TITLE:** **Process control system calls**

**PROBLEM STATEMENT:**

Process control system calls - Fork, execve and wait system calls along with the demonstration of zombie and orphan states.

1. Application should consist of Fork –wait combination (parent with one application and child with another application) and students must demonstrate zombie and orphan states.
2. Application should consist of Fork –execve combination (parent with one application and child with another application).

**THEORY:**

* Wait: When a process develops a child process, it’s occasionally important for the parent process to wait till the child has completed it before continuing. This is exactly what the wait () system function accomplishes. Waiting causes the parent to wait for the child to alter its state. The status change could be due to the child process being terminated, stopped by a signal, or resumed by a signal.
* Zombie Process: A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.
* Orphan Process: A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process. In the following code, parent finishes execution and exits while the child process is still executing and is called an orphan process now

* Process: A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

**PROGRAM:**

#include <stdio.h>

#include<sys/types.h> //fork, sleep, getpid, getppid

#include<sys/wait.h> //system call - wait

#include<stdlib.h>

#include<unistd.h>

int main()

{

pid\_t cpid; //Declaring a variable cpid with the pid\_t data type int \*status=NULL; //Intilizating a pointer var status to NULL

cpid = fork(); //The process fork\_wait creates a new process --clone

if( cpid == 0 ) { //CHILD PROCESS as it is not creating any new process printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* This is child process \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n ");

//write sys call

printf("\n\t Process id is : %d", getpid()); printf("\n\t Parent's process id is : %d", getppid());

sleep(15);

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*Child process terminates \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

else { /\*Parent process waiting for child process, to complete the task\*/

printf("\n\t My process id is : %d", getpid()); printf("\n\t My Parent process id is : %d", getppid()); cpid = wait(status); //Forceful wait; that collects the exit status of child

process with cpid

printf("\n\n\t Parent process collected the exit status of child process with PID %d\n\n", cpid);

}//end of if-else

return 0;

}//end of main

**OUTPUT:**



**CONCLUSION:**

Hence, we have successfully implemented the code for parent-child process to show fork (), wait() functions and understood its concepts theoretically.