**ADVANCED PROCESS ASSIGNMENT**

1. C program to demonstrate zombie process in C.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

int main()

{

    pid\_t child\_id = fork();

    if (child\_id > 0) {

        sleep(50);

        printf("This is parent process.\n");

    }

    else if (child\_id == 0) exit(EXIT\_SUCCESS);

    return 0;

}

**OUTPUT:**



1. C program to demonstrate orphan process in C.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

int main()

{

    pid\_t child\_id = fork();

    if (child\_id > 0) {

        printf("This is parent process.\n");

        printf("Process ID : [%d]\n", getpid());

    }else if (child\_id == 0) {

*//as child process sleeps for 20 seconds, parent process is executed first*

*//child will be executes with init as its parent process*

        sleep(20);

        printf("This is child process.\n");

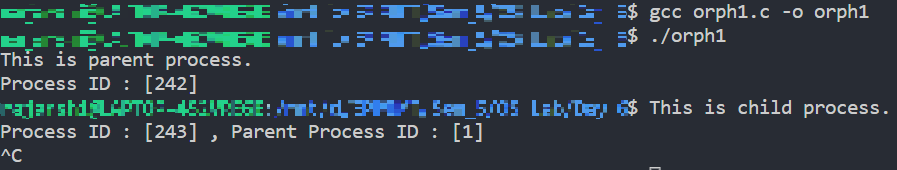
        printf("Process ID : [%d] , Parent Process ID : [%d]\n", getpid(), getppid());

    }

    return 0;

}

**OUTPUT:**



1. C program to demonstrate wait calls for multiple child processes.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <unistd.h>

int main(){

    pid\_t child\_id = fork();

    if (child\_id == 0){

        printf("This is 1st child with Process ID : [%d]\n", getpid());

        sleep(10);

        exit(EXIT\_SUCCESS);

    }

    child\_id = fork();

    if (child\_id == 0){

        printf("This is 2nd child with Process ID : [%d]\n", getpid());

        sleep(10);

        exit(EXIT\_SUCCESS);

    }

    child\_id = wait(NULL);

    printf("1st wait call :: 1st child has exited; with Process ID : [%d]\n", child\_id);

    child\_id = wait(NULL);

    printf("2nd wait call :: 2nd child has exited; with Process ID : [%d]\n", child\_id);

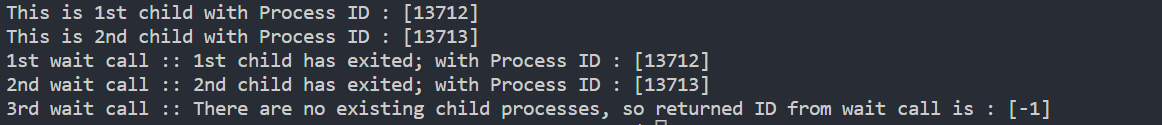
    child\_id = wait(NULL);

    printf("3rd wait call :: There are no existing child processes, so returned ID from wait call is : [%d]\n", child\_id);

    return 0;

}

**OUTPUT:**



Innovative Assignment :

Write a program to create a child process, make the parent process intelligent enough to respawn a new child if somehow the existing child being terminated/killed.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <time.h>

#include <unistd.h>

int main(){

    int m, child\_id;

    child\_id = fork();

    if (child\_id == 0){

        printf("This is first child process\n");

        exit((rand() % 256)+ 1);

    }

    child\_id = wait(&m);

    printf("First child process :: Returned ID : [%d], exit status : [%d]\n", child\_id, WEXITSTATUS(m));

    while(WIFEXITED(m) || WIFSTOPPED(m) || WIFSIGNALED(m)){

        child\_id = fork();

        srand(time(NULL));

        if (child\_id == 0){

            printf("This is spawned child process with Process ID : [%d]\n", getpid());

            if (WIFEXITED(m)) printf("Previous process exited normally :: Exit status of previous child process : [%d]\n", WEXITSTATUS(m));

            if (WIFSTOPPED(m)) printf("Process stopped :: Exit status of previous child process : [%d]\n", WSTOPSIG(m));

            if (WIFSIGNALED(m)) printf("Process exited with exit signal : [%d]\n", WTERMSIG(m));

            sleep(20);

            exit((rand() % 256)+ 1);

        }

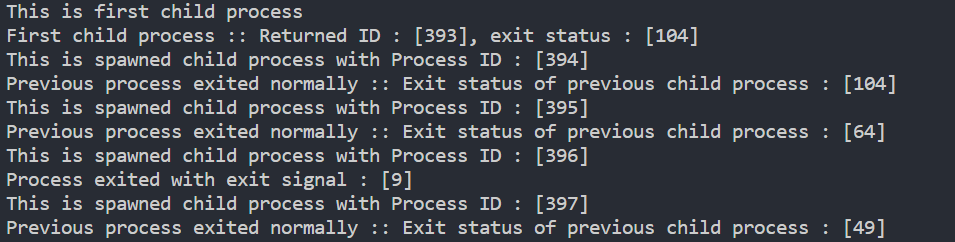
        wait(&m);

    }

    return 0;

}

**OUTPUT:**



Here, we have used SIGKILL to kill the process with PID= 395, hence the exit signal 9

