**A1:Q1**

**Q1) Write a Program to do these: Child process would print next even number, Parent process would print next odd number of a number taken through keyboard in parent process**.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h> // Added for wait()

int main () {

int num;

printf("Enter a number: "); // Input prompt

scanf("%d", &num);

pid\_t pid = fork (); // Creation of child process

if (pid < 0) {// Error handling for fork failure

fprintf(stderr, "Fork failed\n");

return 1;

}

else if (pid == 0) { // Child process

if (num % 2 == 0) {

printf("Child process (PID: %d): Next even number is %d\n", getpid(), num + 2);

} else {

printf("Child process (PID: %d): Next even number is %d\n", getpid(), num + 1);

}

} else { // Parent process

wait(NULL); // Parent waits for child to complete

if (num % 2 == 0) {

printf("Parent process (PID: %d): Next odd number is %d\n", getpid(), num + 1);

} else {

printf("Parent process (PID: %d): Next odd number is %d\n", getpid(), num + 2);

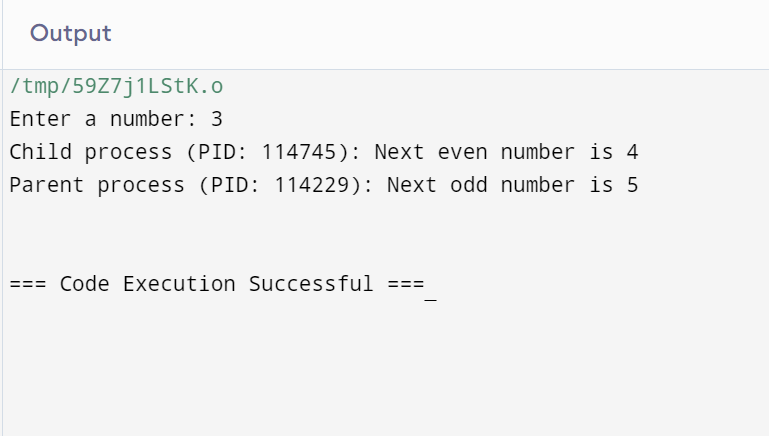
}

}

return 0;

}

**OUTPUT:**



**Explanation:**

**• number: This variable will hold the user-provided input value.**

**● pid: This variable will store the process ID returned by the fork() system call.**

**• The fork() system call is used to create a new child process. As a result, two processes are executed: the parent process and the child process.**

**• If fork() returns a negative value, it indicates that the child process creation failed.**

**• If pid == 0, the code within this block is executed by the child process: ○ If number is even (number % 2 == 0), the program prints the next even number (number + 2).**

**• If number is odd, it prints the next even number (number + 1). ● • If pid > 0, the parent process is executed: ○**

**• The parent process waits for the child process to finish using wait(NULL). ○**

**• Once the child process has completed, the parent checks whether number is even or odd and then prints the corresponding next odd number:**

**• If number is even, it prints the next odd number (number + 1).**

**• If number is odd, it prints the next odd number (number + 2).**

**A1:Q2**

**Q2) Write a program to update the value of variable x by assigning a new constant in the child process.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <unistd.h> // Required for fork ()

int main() {

int x = 10; // initial value of x

printf("Initial value of x: %d\n", x);

pid\_t pid = fork(); //for child process

if (pid < 0) { // for fork failed

fprintf(stderr, "Fork failed\n");

return 1;

} else if (pid == 0) { // for child process

printf("Child process: PID = %d\n", getpid());

x = 20; // gives a new constant value to x in the child process

printf("Child process: Updated value of x = %d\n", x);

} else {// for parent process

printf("Parent process: PID = %d\n", getpid());

// Wait for child process to complete

wait(NULL);

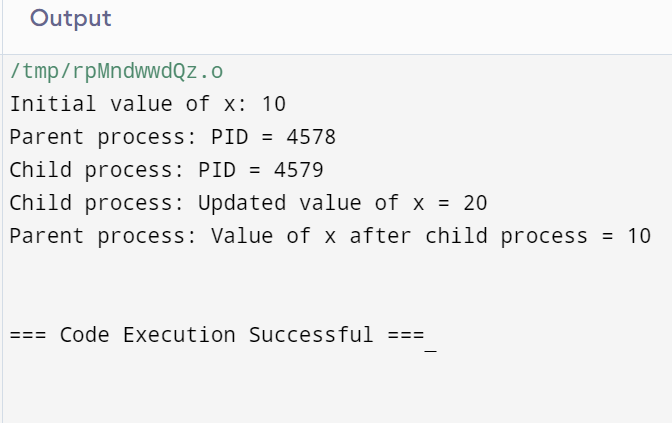
printf("Parent process: Value of x after child process = %d\n", x);

}

return 0;

}

**OUTPUT:**

****

**Explanation:**

**Forking:**

**The fork() system call is used to create a new child process. After forking, both the parent and child processes have their own separate copies of the variable x.**

**Child Process:**

**else if (pid == 0): This block is executed by the child process (when fork() returns 0).**

**● In this block, the child process changes the value of x to a new constant (x = 10;).**

**● It then prints the modified value of x to demonstrate the change.**

**Parent Process:**

**• else: This block is executed by the parent process (when fork() returns a positive value, indicating the child’s process ID).**

**• The parent process waits for the child process to finish using wait(NULL);.**

**• After the child process has completed, the parent process prints the value of x, which remains unaltered in the parent’s context.**