**A2:Q1**

**Q1) WAP to do these: call fork () two times in your main function. Put three printf (“Hello guys”); ---one before the first fork () call, one after the 2nd fork() call and one between the two fork() calls. Each time print the process id along with the “hello guys”. Justify the output with proper explanation.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

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

int main() { // Print before the first fork

printf("Before first fork: Hello guys! PID = %d\n", getpid());

pid\_t pid1 = fork(); // First fork call

printf("Between forks: Hello guys! PID = %d\n", getpid()); // Print between the two fork calls

pid\_t pid2 = fork(); // Second fork call

printf("After second fork: Hello guys! PID = %d\n", getpid()); // Print after the second fork

return 0;}

**OUTPUT:**

****

**Explanation:**

**Before the First Fork:**

**● Only the original parent process executes and prints: Hello guys, before first fork(). PID: 16566. Between Forks:**

**● After the first fork(), two processes exist: the parent and the first child (PID 16567).**

**● Both the parent and the first child process print: Hello guys, between fork() calls. PID: 16566 and Hello guys, between fork() calls. PID: 16567.**

**After the Second Fork:**

**● The second fork() results in the creation of two additional processes, leading to the following:**

**○ The original parent (PID 16566) spawns a new child (PID 1002).**

**○ The first child (PID 16567) spawns its own child (PID 16569).**

**A2:Q2**

**Q2) WAP to do these: call fork() function inside a loop whose number of iteration would be controlled by the user input. For every process print the process id and also print the parent process id of the corresponding process. Justify the output with proper explanation.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h> // Required for pid\_t

#include <unistd.h> // Required for fork(), getpid(), and getppid()

int main() {

int n;

printf("Enter the number of iterations: "); // input for iter

scanf("%d", &n);

for (int i = 0; i < n; i++) {

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

if (pid < 0) {

// Fork failed

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

return 1;

} else if (pid == 0) {

// Child process

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

break; // Prevent child process from continuing the loop

} else {

// Parent process

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

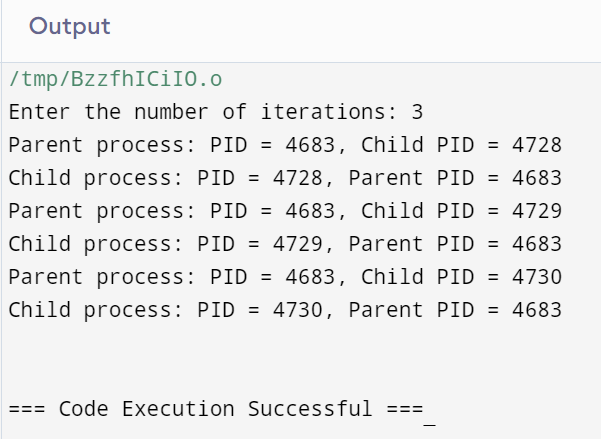
}

}

return 0;

}

**OUTPUT:**

****

**Explanation:**

**Iteration 1:**

**● The first fork() call creates a child process.**

**○ The parent process (PID 2536) prints: Parent Process: PID = 2536, Parent PID = 3564.**

**○ The newly created child process (PID 2565) prints: Child Process: PID = 2565, Parent PID = 1.**

**Iteration 2:**

**● During the second iteration, the parent process performs another fork().**

**○ The parent process (PID 2536) prints: Parent Process: PID = 2536, Parent PID = 3565.**

**○ The new child process (PID 2566) prints: Child Process: PID = 2566, Parent PID = 1.**

**Iteration 3:**

**● In the third iteration, the parent process performs another fork().**

**○ The parent process (PID 2536) prints: Parent Process: PID = 2536, Parent PID = 3566.**

**○ The newly spawned child process (PID 2567) prints: Child Process: PID = 2567, Parent PID = 1**