Name	Manish Shashikant Jadhav
UID no.	2023301005

Experiment 3	
AIM :	Write a program which creates exactly 16 copies of itself by calling fork () only twice within a loop. The program should also print a tree of the pids.

Discussion & Output:	**Fork.c**
	#include <stdio.h>
	#include <sys/wait.h>
	#include<unistd.h>
	
	void printTree(int pid,int level){
	for(int i=0;i<level;i++){
	printf("\t");
	}
	printf("PID:%d\n",pid);
	}
	
	int main()
	{
	int process=16;
	int level=0;
	pid_t parentpid=getpid();
	
	for(int i=0;i<process;i++){
	pid_t childpid=fork();
	
	if(childpid==0){
	level++;
	printTree(getpid(),level);
	}
	else if(childpid>0){
	wait(NULL);
	break;
	}

```
                else{
                        printf("Fork failed.\n");
                        return 1;
                }
        }

        if(getpid()==parentpid){
                for(int i=0;i<process;i++){
                        wait(NULL);
                }
                printf("All processses created.\n");
        }

        return 0;
}
```

Output:

2) Demonstrate the following system calls with examples.

Fork() System call, Wait() system call, Orphan Process, Zombie process.

1. **fork():**

```
#include <stdio.h>

#include <unistd.h>
```

```
int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        printf("Child process: PID = %d\n", getpid());
    } else if (pid > 0) {
        // Parent process
        printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);
    } else {
        // Fork failed
        printf("Fork failed.\n");
        return 1;
    }

    return 0;
}
```

Output:

```
manishj@ubuntu:~/Desktop/osexp3$ gedit fork2.c
^C
manishj@ubuntu:~/Desktop/osexp3$ gcc fork2.c
manishj@ubuntu:~/Desktop/osexp3$ ./a.out
Parent process: PID = 12765, Child PID = 12766
Child process: PID = 12766
manishj@ubuntu:~/Desktop/osexp3$
```

2. **wait():**

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

int main() {
   pid_t childPid;

   // Create a child process
   childPid = fork();

   if (childPid == -1) {
      perror("fork");
      exit(EXIT_FAILURE);
   }

   if (childPid == 0) {
      // Child process
      printf("Child process (PID: %d) is running.\n", getpid());
      sleep(3);  // Simulate some work in the child process
      printf("Child process (PID: %d) is exiting.\n", getpid());
      exit(EXIT_SUCCESS);
   } else {
      // Parent process
      printf("Parent process (PID: %d) is waiting for the child to exit.\n", getpid());
```

```
        int status;
        pid_t terminatedChildPid = wait(&status);

        if (terminatedChildPid == -1) {
            perror("wait");
            exit(EXIT_FAILURE);
        }

        if (WIFEXITED(status)) {
            printf("Child process (PID: %d) exited with status %d.\n", terminatedChildPid,
WEXITSTATUS(status));
        } else if (WIFSIGNALED(status)) {
            printf("Child process (PID: %d) terminated by signal %d.\n", terminatedChildPid,
WTERMSIG(status));
        }

        printf("Parent process (PID: %d) is done.\n", getpid());
    }

    return 0;
}
```

Output:

3. **orphan():**

```
#include <stdio.h>
#include <unistd.h>
```

```
int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        sleep(5);
        printf("Orphan Child process: PID = %d, Parent PID = %d\n", getpid(), getppid());
    } else if (pid > 0) {
        // Parent process
        printf("Parent process: PID = %d\n", getpid());
    } else {
        // Fork failed
        printf("Fork failed.\n");
        return 1;
    }

    return 0;
}
```

Output:

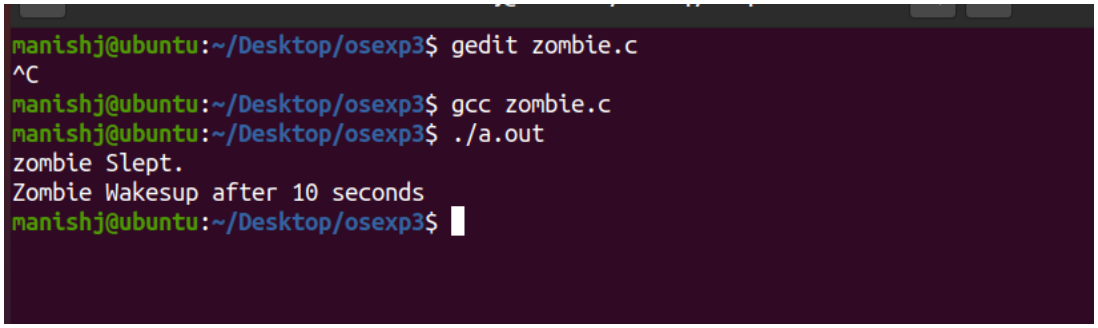
```
manishj@ubuntu:~/Desktop/osexp3$ gedit orphan.c
^C
manishj@ubuntu:~/Desktop/osexp3$ gcc orphan.c
manishj@ubuntu:~/Desktop/osexp3$ ./a.out
Parent process: PID = 12817
manishj@ubuntu:~/Desktop/osexp3$ Orphan Child process: PID = 12818, Parent PID = 1431
^C
manishj@ubuntu:~/Desktop/osexp3$
```

4. **zombie():**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        // Fork failed
        fprintf(stderr, "Fork failed.\n");
```

```
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("zombie Slept.\n");
        exit(0); // Child process exits immediately
    } else {
        // Parent process
        sleep(10); // Parent process sleeps for 5 seconds
        printf("Zombie Wakesup after 10 seconds\n");
    }

    return 0;
}
```

Output:



CONCLUSION: Hence, by completing this experiment I came to know about how to write a program which creates exactly 16 copies of itself by calling fork () only twice within a loop. The program should also print a tree of the pids and also to demonstrate wait() system call, orphan(), and zombie() function.