Operating Systems Lab 4

Name : Jagtap Mahesh

Reg No. 24MCS1017

1) Create a child process from a parent process and print the following:

   From Child:

   Hi I am _____ (Your name)

   I am son/daughter of _____ (Your father or mother name)

   From Parent:

   Hi I am _____ (Your father or mother name)

   I am father/mother of _____ (Your name)

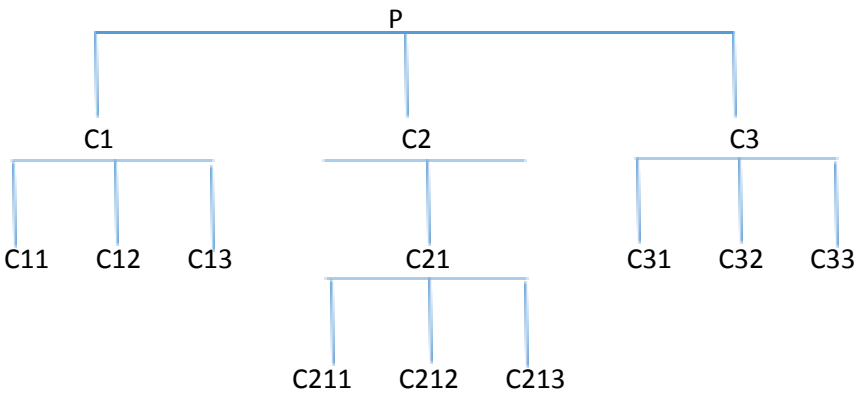
Code:

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

int main() {
   pid_t pid = fork();  // Create a child process

   if (pid < 0) {
      // Fork failed
      printf("Fork failed!\n");
      return 1;
   }
   else if (pid == 0) {
      // Child process
      printf("From Child:\n");
      printf("Hi I am [Your Name]\n");
      printf("I am the son/daughter of [Your Parent's Name]\n");
   }
   else {
      // Parent process
      printf("From Parent:\n");
      printf("Hi I am [Your Parent's Name]\n");
      printf("I am the father/mother of [Your Name]\n");
   }
```

```
    return 0;
}
```

```
$ ./process_example
From Child:
Hi I am Mahesh
I am the son of Vilas
From Parent:
Hi I am Vilas
I am the father of Mahesh
```

2) Create process and its descendants using the following tree



Print the child id along with its parent id.

code:

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

void create_child_process(pid_t parent_pid, const char *child_name) {
    pid_t pid = fork();
```

```
    if (pid == 0) {
        printf("Child: %s, PID: %d, Parent PID: %d\n", child_name, getpid(), parent_pid);
    } else {
        wait(NULL);
    }
}

int main() {
    printf("Parent process (P), PID: %d\n", getpid());
    create_child_process(getpid(), "C1");
    create_child_process(getpid(), "C2");
    create_child_process(getpid(), "C3");

    if (fork() == 0) {
        create_child_process(getpid(), "C11");
        create_child_process(getpid(), "C12");
        create_child_process(getpid(), "C13");
    } else if (fork() == 0) {
        create_child_process(getpid(), "C21");

        if (fork() == 0) {
            create_child_process(getpid(), "C211");
            create_child_process(getpid(), "C212");
            create_child_process(getpid(), "C213");
        }
    } else if (fork() == 0) {
        create_child_process(getpid(), "C31");
        create_child_process(getpid(), "C32");
        create_child_process(getpid(), "C33");
    }
```

```
    while (wait(NULL) > 0);
    return 0;
}
```

```
Parent process (P), PID: 3456
Child: C1, PID: 3457, Parent PID: 3456
Child: C2, PID: 3458, Parent PID: 3456
Child: C3, PID: 3459, Parent PID: 3456
Child: C11, PID: 3460, Parent PID: 3457
Child: C12, PID: 3461, Parent PID: 3457
Child: C13, PID: 3462, Parent PID: 3457
Child: C21, PID: 3463, Parent PID: 3458
Child: C211, PID: 3464, Parent PID: 3463
Child: C212, PID: 3465, Parent PID: 3463
Child: C213, PID: 3466, Parent PID: 3463
Child: C31, PID: 3467, Parent PID: 3459
Child: C32, PID: 3468, Parent PID: 3459
Child: C33, PID: 3469, Parent PID: 3459
```

3) Create a sample program to print an example child process with properties of Zombie and Orphan

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

int main() {
```

```
    pid_t pid = fork();  // Create a child process

    if (pid < 0) {
        // Fork failed
        printf("Fork failed!\n");
        exit(1);
    }

    if (pid == 0) {
        // This is the child process
        printf("Child process: PID = %d, Parent PID = %d\n", getpid(), getppid());
        sleep(5);
        printf("Child process finished (becoming a Zombie if parent hasn't called wait).\n");
    } else {
        // This is the parent process
        printf("Parent process: PID = %d\n", getpid());
        sleep(10);
        printf("Parent process exiting. Child is now an Orphan if still running.\n");
    }
    return 0;
}
```

```
$ gcc zombie_orphan_example.c -o zombie_orphan_example
$ ./zombie_orphan_example
Parent process: PID = 5678
Child process: PID = 5679, Parent PID = 5678
Child process finished (becoming a Zombie if parent hasn't called wait).
Parent process exiting. Child is now an Orphan if still running.
```