1 Write a C program to illustrate that performing ‘n’ consecutive fork() system calls generates a total of $2^n – 1$ child process. The program should prompt the user to input value of ‘n’.

```
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

   int n, i;

   pid_t pid;

   int child_processes = 0;

   // Prompt the user to input the value of n

   printf("Enter the number of fork calls (n): ");

   scanf("%d", &n);

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

      pid = fork();  // Create a new process

      if (pid == 0) {

         // This is the child process

         printf("Child process %d created with PID %d from parent PID %d\n", i + 1, getpid(),
getppid());

         child_processes++;

      } else if (pid > 0) {

         // Parent process continues

         break; // Parent process only forks once and then stops

      } else {

         // fork() failed

         printf("Fork failed\n");

         return 1;

      }
```

```
    }
    // Parent waits for the child processes to finish
    while (wait(NULL) > 0);
    // Output the result
    if (pid > 0) {
        printf("Total number of child processes created: %d\n", (1 << n) - 1);  // 2^n - 1
    }
    return 0;
}
```

2 Write a C program utilizing the fork() system call to generate the following process hierarchy:

P1 -> P2 -> P3. The program should display the Process ID (PID) and Parent Process IDs (PPID) for each process created.

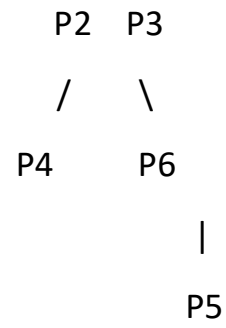
```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
int main() {
    pid_t pid;
    // P1: The parent process starts here
    printf("P1: PID = %d, PPID = %d\n", getpid(), getppid());
    // Create P2: Child of P1
    pid = fork();
    if (pid < 0) {
        // fork failed
        printf("Fork failed\n");
        exit(1);
```

```
    }
    if (pid == 0) {
        // P2: This is the child process of P1
        printf("P2: PID = %d, PPID = %d\n", getpid(), getppid());
        // Create P3: Child of P2
        pid = fork();
        if (pid < 0) {
            // fork failed
            printf("Fork failed\n");
            exit(1);
        }
        if (pid == 0) {
            // P3: This is the child process of P2
            printf("P3: PID = %d, PPID = %d\n", getpid(), getppid());
        } else {
            // P2 waits for P3 to finish
            wait(NULL);
        }
    } else {
        // P1 waits for P2 to finish
        wait(NULL);
    }
    return 0;
}
```

3 Write a C program to generate a process hierarchy as follows:

P1

/ \



The program should create the specified process structure using the appropriate sequence of fork() system calls. Print PID and PPID of each process.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
int main() {
   pid_t pid;
   // P1: The parent process starts here
   printf("P1: PID = %d, PPID = %d\n", getpid(), getppid());
   // Create P2: Child of P1
   pid = fork();
   if (pid < 0) {
      // fork failed
      printf("Fork failed\n");
      exit(1);
   }
   if (pid == 0) {
      // P2: This is the child process of P1
      printf("P2: PID = %d, PPID = %d\n", getpid(), getppid());
      // Create P4: Child of P2
      pid = fork();
```

```
    if (pid < 0) {
        // fork failed
        printf("Fork failed\n");
        exit(1);
    }
    if (pid == 0) {
        // P4: This is the child process of P2
        printf("P4: PID = %d, PPID = %d\n", getpid(), getppid());
        // Create P5: Child of P4
        pid = fork();
        if (pid < 0) {
            // fork failed
            printf("Fork failed\n");
            exit(1);
        }
        if (pid == 0) {
            // P5: This is the child process of P4
            printf("P5: PID = %d, PPID = %d\n", getpid(), getppid());
        } else {
            // P4 waits for P5 to finish
            wait(NULL);
        }
    } else {
        // P2 waits for P4 to finish
        wait(NULL);
    }
} else {
```

```
// Create P3: Another child of P1
pid = fork();
if (pid < 0) {
    // fork failed
    printf("Fork failed\n");
    exit(1);
}
if (pid == 0) {
    // P3: This is the second child process of P1
    printf("P3: PID = %d, PPID = %d\n", getpid(), getppid());
    // Create P6: Child of P3
    pid = fork();
    if (pid < 0) {
        // fork failed
        printf("Fork failed\n");
        exit(1);
    }
    if (pid == 0) {
        // P6: This is the child process of P3
        printf("P6: PID = %d, PPID = %d\n", getpid(), getppid());
    } else {
        // P3 waits for P6 to finish
        wait(NULL);
    }
} else {
    // P1 waits for both P2 and P3 to finish
    wait(NULL);
```

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