Q1)write a c program to create the process heirarchy
p0->p1->p2->p3->p4->p5.Where p0 is the original process

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

void createProcessHierarchy(int depth) {
    if (depth == 0) return;

    pid_t pid = fork();

    if (pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Process p%d (PID: %d, Parent PID: %d)\n", 6 - depth, getpid(), getppid());
        createProcessHierarchy(depth - 1);
    } else {
        // Parent process
        wait(NULL);
    }
}

int main() {
    printf("Process p0 (PID: %d)\n", getpid());
    createProcessHierarchy(5);
    return 0;
}
```

Q2)Develop a c prog to create a chain of n(n>4) process.The value of n will be given from command line . Display the pid ppid and the return value of the fork() for n no of processes.Print the pid and ppid such that eaach child will be completed before its parents are termined

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

void createChain(int depth) {
    if (depth == 0) return;

    pid_t pid = fork();

    if (pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child process %d -> pid: %d, ppid: %d, fork return value: %d\n", depth, getpid(), getppid(), pid);
        createChain(depth - 1);
    } else {
        // Parent process
        wait(NULL); // Wait for child to complete
        printf("Parent process -> pid: %d, ppid: %d, fork return value: %d\n", getpid(), getppid(), pid);
        exit(EXIT_SUCCESS);
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <n>\n", argv[0]);
        return EXIT_FAILURE;
    }

    int n = atoi(argv[1]);

    if (n <= 4) {
        fprintf(stderr, "Error: n must be greater than 4\n");
        return EXIT_FAILURE;
    }
    createChain(n);

    return EXIT_SUCCESS;}
```

Q) Develop a C program to create two processes, **P1** and **P2**, using fork(). Ensure that neither process becomes an orphan. The two processes must communicate using semaphores to synchronize their outputs.

- **P1** should print the message: PPWC
- **P2** should print the message: DOS

The output must follow the specific alternating sequence:
DOS PPWC DOS PPWC ...

CODE:

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

int main() {
  // Create two named semaphores
  sem_t *sem1 = sem_open("/sem1", O_CREAT, 0644, 0); // Semaphore for P1
  sem_t *sem2 = sem_open("/sem2", O_CREAT, 0644, 1); // Semaphore for P2

  if (sem1 == SEM_FAILED || sem2 == SEM_FAILED) {
    perror("Semaphore initialization failed");
    exit(EXIT_FAILURE);
  }

  pid_t pid1 = fork();

  if (pid1 == 0) {
    // Child process P1
    while (1) {
      sem_wait(sem1); // Wait for P2 to print DOS
      printf("PPWC\n");
      fflush(stdout);
      sem_post(sem2); // Signal P2 to print DOS
      sleep(1);
    }
  } else {
    pid_t pid2 = fork();

    if (pid2 == 0) {
      // Child process P2
      while (1) {
        sem_wait(sem2); // Wait for P1 to print PPWC
        printf("DOS\n");
        fflush(stdout);
        sem_post(sem1); // Signal P1 to print PPWC
        sleep(1);
      }
    } else {
      // Parent process: Wait for both child processes to finish
      wait(NULL);
      wait(NULL);

      // Cleanup semaphores
      sem_unlink("/sem1");
      sem_unlink("/sem2");
    }
  }

  return 0;
}
```

Q) Create three process p1,p2,p3 such that there will be no orphan and each process will display a message as p1-who and p2-are and p3-you .develop a program using semaphore to display the message in the given order

CODE:

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

int main() {
```

```
    // Create named semaphores
    sem_t *sem1 = sem_open("/sem1", O_CREAT, 0644, 0); // Semaphore for P1
    sem_t *sem2 = sem_open("/sem2", O_CREAT, 0644, 0); // Semaphore for P2
    sem_t *sem3 = sem_open("/sem3", O_CREAT, 0644, 1); // Semaphore for P3 (Initially allows P3 to run)

    if (sem1 == SEM_FAILED || sem2 == SEM_FAILED || sem3 == SEM_FAILED) {
        perror("Semaphore initialization failed");
        exit(EXIT_FAILURE);
    }

    pid_t pid1 = fork();

    if (pid1 == 0) {
        // Child process P1
        printf("p1-who\n");
        fflush(stdout);
        sem_post(sem1); // Signal P2 to run
        sem_wait(sem3); // Wait for P3 to finish
        exit(0);
    } else {
        pid_t pid2 = fork();

        if (pid2 == 0) {
            // Child process P2
            sem_wait(sem1); // Wait for P1 to print
            printf("p2-are\n");
            fflush(stdout);
            sem_post(sem2); // Signal P3 to run
            exit(0);
        } else {
            // Parent process (P3)
            sem_wait(sem2); // Wait for P2 to print
            printf("p3-you\n");
            fflush(stdout);
            sem_post(sem3); // Allow P1 to finish and exit

            // Wait for child processes to finish
            wait(NULL);
            wait(NULL);

            // Cleanup semaphores
            sem_unlink("/sem1");
            sem_unlink("/sem2");
            sem_unlink("/sem3");
        }
    }

    return 0;
}
```

Q)
A)Write a code segment to initialise a semaphore 5 to 10 and display the semaphore value.
CODE:

```
#include <stdio.h>
#include <semaphore.h>
#include <fcntl.h> // For O_CREAT
#include <sys/stat.h> // For permissions

int main() {
    // Initialize the semaphore with a value between 5 and 10 (e.g., 5)
    int semaphore_value = 5; // Set this to any value between 5 and 10

    // Create or open a named semaphore
    sem_t *semaphore = sem_open("/my_semaphore", O_CREAT, 0644, semaphore_value);

    // Check if semaphore was created successfully
    if (semaphore == SEM_FAILED) {
        perror("sem_open failed");
        return 1;
    }

    // Display the initialized semaphore value (for demonstration purposes)
    printf("Semaphore initialized with value: %d\n", semaphore_value);

    // Close the semaphore (not removing it yet)
    sem_close(semaphore);

    return 0;
```

}

B) Following the previous question the operation applied on semaphore as sem_wait(s) again sem_sem (s) next sem_post(s) then sem_post(s) and sem_wait(s).find the value of the semaphore.

- Initial value: 5
- First sem_wait(s): 5 → 4
- Second sem_wait(s): 4 → 3
- First sem_post(s): 3 → 4
- Second sem_post(s): 4 → 5
- Final sem_wait(s): 5 → 4

Thus, after applying the sequence of operations, the semaphore value is **4**.

C Code Examples: Orphan Process and `ls -l process.c` Execution

1. Orphan Process and Prevention

The following code demonstrates how an orphan process is created and how to modify the code to prevent it.

Orphan Process Code:

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

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

   if (pid < 0) {
      // Fork failed
      perror("Fork failed");
      exit(1);
   } else if (pid == 0) {
      // Child process
      printf("Child process: PID = %d, Parent PID = %d\n", getpid(), getppid());
      sleep(5); // Let the child process run for a while
      printf("Child process: PID = %d, Parent PID = %d (after parent exit)\n", getpid(), getppid());
   } else {
      // Parent process
      printf("Parent process exiting: PID = %d\n", getpid());
      exit(0); // Parent process exits immediately
   }

   return 0;
}
```

Orphan Prevention Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h> // For wait()

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

   if (pid < 0) {
      // Fork failed
      perror("Fork failed");
      exit(1);
   } else if (pid == 0) {
      // Child process
      printf("Child process: PID = %d, Parent PID = %d\n", getpid(), getppid());
      sleep(5); // Simulate work done by child
      printf("Child process completed: PID = %d\n", getpid());
   } else {
      // Parent process
      printf("Parent process waiting for child to complete: PID = %d\n", getpid());
      wait(NULL); // Wait for the child process to terminate
      printf("Parent process: Child has terminated, now exiting.\n");
   }

   return 0;
}
```

2. `ls -l process.c` Execution in a Child Process
The following code creates a child process using `fork()`, executes the `ls -l process.c` command using `exec()` in the child process, and waits for the child to terminate in the parent process.

`ls -l process.c` Execution Code:

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

int main() {
   pid_t pid; // Variable to store process ID
   int status; // Variable to store status of the child process

   // Fork a child process
   pid = fork();

   if (pid < 0) {
      // Fork failed
      perror("Fork failed");
      exit(1);
   } else if (pid == 0) {
      // Child process
      printf("Child process: Executing 'ls -l process.c'\n");
      // Execute the `ls -l process.c` command
      execl("/usr/bin/ls", "ls", "-l", "process.c", NULL);
      // If exec() fails, print an error and exit
      perror("execl failed");
      exit(1);
   } else {
      // Parent process
      printf("Parent process: Waiting for child process to terminate...\n");
      pid_t child_pid = wait(&status); // Wait for the child to finish
      if (WIFEXITED(status)) {
         printf("Parent process: Child with PID %d terminated successfully with exit code %d.\n", child_pid, WEXITSTATUS(status));
      } else {
         printf("Parent process: Child with PID %d terminated abnormally.\n", child_pid);
      }
   }

   return 0;
}
```