A) (2 points)

If you create a main() routine that calls `fork()` twice, followed by a call to `execl()`, i.e. if it includes the following code:

```
pid_t x=-11, y=-22;
x = fork();
if(x==0) y = fork();
if(y>0) execl("/bin/ls", "ls", "-l", NULL);
```

Assuming all `fork()` calls succeed, and `execl` replaces the process image, draw a process tree similar to that of Fig. 3.8 (page 116) in your textbook, clearly indicating the values of x and y for each process in the tree (i.e. whether 0, -11, -22, or larger than 0). Note that the process tree should only have one node for each process and thus the number of nodes should be equal to the number of processes. The process tree should be a snapshot just after all forks completed but before any process exits (except for the process that executed execl).

B) (4 points)

Write a program that creates the following process tree:

```
       P1
      /  \
     P2   P3
    / \    |
   P4 P5  P6
```

Where P1 is the parent process, and P2, P3, P4, P5, P6 are its children and grandchildren. Each process should print its process ID.

C) (4 points)

Write a program whose main routine obtains two parameters from the user (a and b, passed as command-line arguments). The program should create a child process. The child process computes and prints the greatest common divisor (GCD) of a and b using Euclid's algorithm. The parent process waits for the child to finish and then prints the least common multiple (LCM) of a and b, calculated using the relationship LCM(a,b) = (a*b)/GCD(a,b).

D) (3 points)

Explain the difference between `fork()` and `vfork()`. What are the potential dangers of using `vfork()`? Why is `vfork()` often considered obsolete?

E) (3 points)

Write a C program that uses signals to handle a SIGINT (Ctrl+C) signal. When the signal is received, the program should gracefully terminate and print a message indicating it's exiting.

F) (4 points)

Write a program that creates two child processes. The first child process prints the even numbers from 2 to 20. The second child process prints the odd numbers from 1 to 19. The parent process waits for both children to complete before exiting.

G) (4 points)

Write a C program that creates a child process. The parent process sends a message to the child process using a pipe. The child process receives the message, reverses it (character by character), and sends the reversed message back to the parent process. The parent process then prints the reversed message. Ensure proper error handling for pipe creation and operations.