**For Ques1\_1.c**:

This code demonstrates the use of fork() to create a child process. The key points are:

- fork() creates a child process that is a duplicate of the parent. It returns the child's PID to the parent, and 0 to the child.

- After the fork, both the parent and child execute the code below the fork. They can tell whether they are the parent or child based on the return value of the fork().

- The parent prints "(A) Parent (P) is having ID" with its PID.

- The child prints "(C) Child is having ID" with its PID.

- The child also prints "(D) My ID is" with its PID again.

- The parent calls wait() to wait for the child to finish before printing "(B) ID of P's Child is" with the parent's PID again.

- Note that the child finishes before the parent due to the wait(), so the child's prints happen before the parent's last print.

The key point is that fork() creates a separate child process, which allows concurrent execution of parent and child.

**For Ques1\_2.c:**

This code uses vfork() instead of fork(). The key differences are:

- vfork() does not fully duplicate the parent process. It shares memory and resources until the child either exec()s or exits.

- Control is not returned to the parent until the child exits or exec()s. So the child runs first.

- The child prints the factorial calculation.

- The parent prints the Fibonacci series.

- The child explicitly waits for the parent using a print statement.

The key points are that vfork() avoids duplicating resources, but the parent is suspended until the child exits/exec()s. Also, the child runs first before the parent.