1.1.
q1.1.c creates a simple parent process that forks a child process, and both processes print their respective process IDs and parent process IDs.

It uses the fork() system call to create a new process (child) that is a copy of the current process (parent). The return value of fork() is stored in the variable rc.

- If rc is negative, an error occurred during the fork, and an error message is printed to the standard error stream.
- If rc is 0, the process is the child process.
- If rc is positive, the process is the parent process, and the value of rc represents the process ID of the child process.

Inside the else block, the code is executed by the parent process. It waits for the child process to complete using the wait(NULL) system call, which suspends the parent process until the child process terminates. The return value of wait() (child's exit status) is stored in the variable wc.

Inside the if (rc == 0) block, the code is executed by the child process. The child process prints its own process ID using getpid() and its parent's process ID using getppid().

The parent process then prints its own process ID using getpid(), the process ID of the child (stored in rc), and a newline for formatting.

//Write about makefile

all: The default target. When you simply run make, it will execute the run target, which will build and run the program.
q1.1: This target compiles the q1.1.c source file using the gcc compiler and creates an executable named q1.1.
run: This target depends on q1.1. It ensures that the program is built and then runs the executable using ./q1.1.
clean: This target removes the compiled executable q1.1.

1.2.
q1.2.c demonstrates the concept of a child process waiting for its parent process to complete. The program uses the fork() system call to create a child process, and then the child process waits for its parent process to finish execution before proceeding.

p1.c calculates the factorial of a given number using a recursive function named factorial. The recursive approach iterates down to the base case where n is 0 or 1 and returns 1. For other values of n, it multiplies n with the result of the function called with n - 1. It then prints the calculated factorial of the provided number.

p2.c calculates and prints the Fibonacci sequence using a recursive function named fibonacci. It uses a recursive approach to calculate the Fibonacci numbers by summing up the results of the function calls for n - 1 and n - 2. The print_fibonacci_sequence function iteratively prints the Fibonacci sequence up to a given number n.
We have considered the fibonacci sequence to start from 0, thus it goes till 610.

//Write about makefile

p1: Compiles the first program p1.c into an executable named p1.
p2: Compiles the second program p2.c into an executable named p2`.
run: Runs q1.2.c.
clean: Removes the compiled executables (p1 and p2).

2.
source.c implements three custom shell commands (word, dir, and date). These commands provide the following functionalities:

1.word command:

The word command is an internal command that calculates the number of words in a text file. It supports options to modify its behavior: -n to ignore newline characters and -d to compare word counts between two files.(Only one of these options can be used at a time.)

It uses the getopt function to parse the command-line options and arguments. The word_count function reads and counts words in a file, and the word_count_difference function calculates the difference in word counts between two files.

2.dir command:

The dir command is an external command that creates a directory and changes the shell's working directory to that directory.
It supports options: -r to remove an existing directory and -v to print messages for each step. (Only one of these options can be used at a time.)

It uses the getopt function to parse the command-line options and arguments. It uses the mkdir function to create a directory and the chdir function to change the working directory. The -r option checks for an existing directory and removes it before creating a new one if specified.

3.date command:

The date command is an external command that retrieves the last modified date and time of a file. It supports options: -d to display time in a specific format and -R to output in RFC 5322 format.(Only one of these options can be used at a time.)

It uses the getopt function to parse the command-line options and arguments. It uses the stat function to retrieve file information, and the localtime and strftime functions to format and display the date and time.

Integration with Custom Shell:

The main source file integrates these commands into a custom shell. When the user inputs a command, the shell processes the input and executes the corresponding command's functionality. The executeExternalCommand function uses fork and exec to run external commands within the shell.

3.
The Bash script creates an interactive calculator that performs three different operations (xor, product, compare) based on input values read from an input file. It stores the results in an output file. Additionally, a Makefile is included to simplify the execution and cleaning of the script.

- The script defines three functions (`xor`, `product`, `compare`) to perform the specified operations on given inputs.

- The script enters a loop that reads three space-separated values (`a`, `b`, `s`) from an input file (`input.txt`).
  - `a` and `b` represent the operands for the calculations.
  - `s` specifies the operation to be performed.

- Depending on the value of `s`, the script calls the corresponding function and stores the result in a variable `c`.

- After each operation, the script prints the values of `a`, `b`, and `s`, along with the calculated result `c`, to the console. It also appends the result to an output file (`output.txt`).

The Makefile enhances the execution process:
- `run`: Compiles the script (if necessary), grants execute permissions, and runs the script with input from `input.txt`.
- `clean`: Removes the compiled script.