Exercise problems on fcntl() System call

1. **File Locking**:
Use the `fcntl` system call to implement a simple file locking mechanism.
Write a C program that takes a file name as a command-line argument and allows two processes to lock and unlock the same file concurrently using `fcntl`.
Ensure that one process cannot modify the file while it is locked by another process.

2. **File Descriptor Duplication**:
Write a C program that opens a file and then duplicates the file descriptor using `fcntl`.
Create two file descriptors that refer to the same open file description.
Test if changes made through one descriptor affect the other.

3. **Set Non-blocking Mode**:
Implement a program that opens a file in non-blocking mode using `fcntl`.
Read data from the file descriptor, and when no data is available, handle the EAGAIN/EWOULDBLOCK error and continue processing.

4. **File Descriptor Flags**:
Create a program that opens a file and retrieves the flags associated with the file descriptor using `fcntl`. Modify the flags to include or exclude O_APPEND and O_NONBLOCK, and then set the updated flags using `fcntl`.

5. **File Locking with Timeout**:
Modify the file locking program from problem 1 to include a timeout mechanism.
If a process attempts to acquire a lock and it cannot do so within a specified time limit, it should return an error. Implement this using `fcntl` and a signal handler for timeout handling.

6. **Record Locking**:
Extend the file locking program to perform record-level locking instead of file-level locking.
Create a program that allows multiple processes to lock and unlock specific records within a file. Use `fcntl` to manage record locks efficiently.

7. **Check for File Lock**:
Write a C program that checks if a given file is locked by another process.
Provide a command-line argument with the file name and use `fcntl` to determine if the file is currently locked. Print a message indicating whether the file is locked or not.

8. **Asynchronous I/O with `fcntl`**:
Create a program that performs asynchronous I/O using the `fcntl` system call.
Open a file and use `fcntl` to set it up for asynchronous reads and writes. Implement a mechanism to read and write data to the file asynchronously and handle the completion of I/O operations.

9. **Change File Descriptor Ownership**:
Write a program that changes the ownership of a file descriptor to another process using `fcntl`. Demonstrate how one process can pass a file descriptor to another process, allowing the second process to access the same file.

10. **File Descriptor Flags and File Status Flags**:
Write a program that opens a file and uses `fcntl` to retrieve both the file descriptor flags and the file status flags associated with the file. Print the values of these flags and explain their meanings.