**Distributed System Lab 4.**

**4.1.** Configure following options on server socket and tests them: SO\_KEEPALIVE, SO\_LINGER, SO\_SNDBUF, SO\_RCVBUF, TCP\_NODELAY.

**Explanation:** SO\_KEEPALIVE: SO\_KEEPALIVE is a socket option that allows the operating system to automatically send keepalive packets on an idle TCP connection to check if the other end is still responsive. In the code, we set SO\_KEEPALIVE to 1 (enabled) using setsockopt. This option helps detect and close inactive connections more reliably.

**4.2**. **Incrementing a counter in shared memory.**

**Explanation**: Incrementing a counter in shared memory refers to the process of increasing the value of a counter variable that is stored in a region of memory that is accessible to multiple threads or processes concurrently. Shared memory is a form of inter-process communication (IPC) that allows different processes or threads to access and manipulate the same memory region, making it a useful mechanism for synchronization and coordination in concurrent programming. Here's a general overview of how incrementing a counter in shared memory might work:

**Initialization:** Initially, you would create or allocate a block of shared memory that can hold the counter variable. This shared memory block is typically created by one process or thread and then shared among multiple processes or threads.

**Access Control:** To ensure that multiple processes or threads don't try to access and modify the shared counter simultaneously (which could lead to race conditions and data corruption), synchronization mechanisms like mutexes, semaphores, or atomic operations are often used. These mechanisms provide a way to control access to the shared memory region.

**Increment Operation:** When a process or thread wants to increment the counter, it first acquires the necessary synchronization mechanism to gain exclusive access to the shared memory. Once it has exclusive access, it can safely read the current value of the counter, increment it, and write the updated value back to the shared memory location.