1.

1. If the malicious process reads the message queue intended for server and clients, it breaks the order of request and response of the server and clients, which confuses the communication between server and client processes.

2. System V message queues:

The malicious process should know the unique key identifier of the queue and the queue must be created with universal read permission.

POSIX message queues:

The malicious process should know the queue name and queue must have read permission enabled.

2.

FIFO:

- Data is in byte streams
- It can be used for communication between unrelated processes if fifo name is shared.
- It can be opened as normal file, and when created it has an entry in file descriptor table.
- Data are read in the same order as written. Hence the name FIFO.
- Once created, any process can open it with permission check.
- Can use it only in BLOCK mode.
- There is no notification registration like Message queue.

POSIX Message Queue

- Data is in message streams
- It can be used for communication between unrelated processes if the msg_queue name is shared.
- When created, it has an entry in message queue descriptor table.
- Message are received strictly in priority order.
- Once created, any process can open it with permission check.
- Can be used either in BLOCK mode or ASYNC mode.
- Can register to receive notification if new message is received.

3.

Binary Semaphores:

- It is a signaling mechanism to notify other thread about an action.
- It can be used to synchronize access to shared memory.
- It can be used between related processes or threads.
- It can be used to synchronize any action between processes.
- Semaphore values are kept in table stored in kernel memory.
- It can act as pthread_mutexes.

Pthread_mutexes:

- It is a shared resource locking mechanism.
- It is used only to synchronize access to shared global memory.

- It can be used only between threads.
- It cannot act as Binary semaphores.
- Mutex can be released only by the thread that had acquired it.

4.

1) Both the program uses SYSTEM V shared memory and Binary semaphore to transfer data from STDIN to shared memory and from shared memory to STDOUT or implementing pipe behavior.

2) After writing all data to shared memory, the writer program reserves the semaphore to confirm whether the reader has read all data from shared memory and remove the shared memory once the semaphore is released by the reader.

3) If reader tries to read from sharedMemory before writer has written into sharedMemory, reader will get EOF as output and shmp->cnt will be zero. Keeping shmp->cnt inside reserveSem guarantees that some data has been written by writer. So shmp->cnt cannot be moved outside reserveSem as initially it will be zero.

5.

1) From line 25 and 29, we can tell that the program is receiving notification via thread. The program is registering for notification via thread and blocking the execution until notification is received.

2) After a notification is delivered, the notification registration is removed. So to register for another notification, it is called again before reading the unread msg.

3) Since SIGEV_THREAD is specified, the threadFunc will be executed as start function of a new thread. Let's assume buffer in threadFunc is made as global and memory is allocated at once. If Multiple message notification is received, multiple threads will be created and will try to access the same buffer object. This will lead to data race condition. So it is not advisable to make the buffer a global variable.

6.

1) If listen() is removed, the accept() in line 22 will return -1 with error status EINTVAL and it will lead to infinite loop.

2) If bind() is removed, then listen() will create an ephemeral port for the server and will listen to that port for connections.

7.

1) Invalid – When pthread_exit() is used within the thread, the process-shared resources are not released. So using pthread_join() or pthread_detach() does not make difference.

2) Invalid – The target thread can be cancelled immediately only if its cancel state is set to PTHREAD_CANCEL_ENABLE and cancel type as PTHREAD_CANCEL_ASYNCHRONOUS or reach a cancellation point if its cancel type is set as PTHREAD_CANCEL_DEFERRED.

3) Invalid – Unnamed POSIX semaphores can be used between threads and related process. It can be used between unrelated processes until the semaphore is in the shared memory region.

4) Invalid – The connect() function can be used for any socket communication. It is not specific to TCP connections.

5) Valid – If two process attach a shared memory segment, then there is no chance that both of them can have same addresses in their virtual address-space.