התקדמות משאלה 4 ל6:

How Reactor Improves Handling of Multiple Clients (Comparison):

Aspect	Exercise 4 (Blocking I/O)	Exercise 6 (Reactor Pattern with Select)
I/O Handling	**Blocking**: Server waits for a client to send data.	**Non-blocking**: Server handles multiple clients without waiting.
Handling Multiple Clients	**Sequential**: One client at a time, others wait.	**Concurrent**: Multiple clients processed concurrently.
Scalability	**Limited**: Difficult to scale to many clients due to blocking.	**Scalable**: Can handle many clients efficiently using non-blocking I/O.
Responsiveness	**Slower**: Client interactions delayed by blocking.	**Faster**: Clients processed as soon as they send data.
Efficiency	**Low**: Wasted CPU cycles waiting for I/O.	**High**: CPU is efficiently used, waiting only for ready FDs.
Architecture	**Synchronous**: One client at a time.	**Event-Driven**: Server reacts to client events (data ready).

Example of How Reactor Helps:

Let's say the server has three connected clients:

- **Client 1** sends the command `Newgraph 5,3`.
- **Client 2** sends the command `Newedge 1,2`.
- **Client 3** sends the command `Kosaraju`.

In **exercise 4**:

- The server would be blocked, waiting for Client 1's command to be fully received and processed before handling Client 2 or Client 3.
- If Client 1 takes time to send data, the server cannot move on to the other clients, leading to inefficiency.

In **exercise 6 (with the reactor)**:

- The reactor adds all client file descriptors (FDs) to the `select()` call.
- The server can **immediately detect** when any client sends data. If Client 1 starts sending data but pauses, the server can still process commands from Client 2 and Client 3 **without waiting**.
- The server is always ready to handle multiple clients concurrently, improving responsiveness and throughput.