Mini-project: Non-blocking I/O and poll()

1. Project Goal:

In this mini-project, you will build a simple server that can handle multiple client connections simultaneously without blocking the server. The goal is to use the poll() function, which allows you to manage multiple socket connections at the same time.

2. Important Functions:

From the previous exercise:

- socket(): Creates a socket.
- bind(): Binds a socket to a specific port and IP address.
- listen(): Puts the socket in a state where it listens for incoming connections.
- accept(): Accepts an incoming connection from a client.
- recv() and send(): Used for receiving and sending data with clients.

New Functions for this exercise:

1. fcntl() (new):

- Used to set the socket to non-blocking mode. This means that socket operations such as accept(), recv(), and send() will not block the program but will return immediately, even if no data is available.

Syntax: fcntl(sockfd, F_SETFL, O_NONBLOCK);

2. poll() (new):

- This is the core function of this project. The poll() function is used to monitor multiple socket connections at the same time. It checks which sockets are ready to perform certain operations (e.g.,

accept(), recv(), send()).

Syntax: int poll(struct pollfd fds[], nfds_t nfds, int timeout);

- fds[]: Array of structures representing the sockets and their events.

- nfds: Number of sockets being monitored.

- timeout: How long poll() should wait before returning (0 for no wait, -1 for infinite wait).

3. struct pollfd (new):

- This structure holds information about one socket being monitored. For each socket you want to track, you define one pollfd structure.

Structure:

```
struct pollfd {
    int fd;        // The socket file descriptor
    short events;  // The events you want to monitor (e.g., POLLIN, POLLOUT)
    short revents; // The events that occurred (set by poll())
};
```

4. New Events:

- POLLIN: There is data to read from the socket.

- POLLOUT: The socket is ready to send data.

3. Project Structure and Steps:

Step 1: Create sockets and set them to non-blocking mode using the fcntl() function.

Step 2: Add the pollfd structure for each socket you want to monitor.

Step 3: Use poll() to monitor events on these sockets.

Step 4: Accept new connections if poll() detects activity on the listening socket.

Step 5: Use recv() and send() to receive and send data if poll() detects activity on client sockets.

Step 6: Close sockets when they are no longer needed.