**CONCURRENT CLIENT SERVER FOR INTEGER ARRAY**

S CODE #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/wait.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

void sort\_array(int arr[], int n) {

int temp;

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (arr[j] > arr[j+1]) {

// Swap

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

int n, arr[BUFFER\_SIZE];

char buffer[BUFFER\_SIZE] = {0};

// Create socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Bind the socket to the network

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

while (1) {

// Accept a new connection

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

// Fork a new process for each client

if (fork() == 0) {

// Child process handles the client

close(server\_fd); // Child process doesn't need the server socket

// Read the size of the array from the client

read(new\_socket, &n, sizeof(int));

// Read the array from the client

read(new\_socket, arr, sizeof(int) \* n);

// Sort the array

sort\_array(arr, n);

// Prepare the response with the sorted array and the process ID (PID)

char response[BUFFER\_SIZE] = {0};

sprintf(response, "Sorted array: ");

for (int i = 0; i < n; i++) {

char num\_str[10];

sprintf(num\_str, "%d ", arr[i]);

strcat(response, num\_str);

}

sprintf(response + strlen(response), "| Server PID: %d", getpid());

// Send the sorted array and PID back to the client

send(new\_socket, response, strlen(response), 0);

// Close the client socket

close(new\_socket);

// Terminate the child process

exit(0);

} else {

// Parent process continues to accept new connections

close(new\_socket); // Parent doesn't need the client socket

}

// Reap zombie processes

while (waitpid(-1, NULL, WNOHANG) > 0);

}

return 0;

}

C CODE #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

int arr[BUFFER\_SIZE];

char buffer[BUFFER\_SIZE] = {0};

int n;

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 address from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

printf("\nInvalid address/ Address not supported \n");

return -1;

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

printf("\nConnection Failed \n");

return -1;

}

// Get the size of the array from the user

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

// Get array elements from the user

printf("Enter %d elements:\n", n);

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

// Send the array size to the server

send(sock, &n, sizeof(int), 0);

// Send the array to the server

send(sock, arr, sizeof(int) \* n, 0);

// Read the sorted array from the server

read(sock, buffer, BUFFER\_SIZE);

printf("Sorted array and PID from server: %s\n", buffer);

// Close the socket

close(sock);

return 0;

}