**TCP DAYTIME SERVER USING SELECT()**

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/time.h>

#include <time.h>

#include <errno.h> // Include this for errno and EINTR

#define PORT 8080

#define MAX\_CLIENTS 30

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket, client\_socket[MAX\_CLIENTS], max\_clients = MAX\_CLIENTS, activity, valread, sd;

int max\_sd;

struct sockaddr\_in address;

fd\_set readfds;

char buffer[BUFFER\_SIZE];

socklen\_t addrlen = sizeof(address);

// Initialize all client\_socket[] to 0, so that they are not checked

for (int i = 0; i < max\_clients; i++) {

client\_socket[i] = 0;

}

// Create a master socket (server socket)

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

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Prepare the sockaddr\_in structure for binding the socket

address.sin\_family = AF\_INET;

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

address.sin\_port = htons(PORT);

// Bind the socket to the network

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

perror("Bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

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

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

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

while (1) {

// Clear the socket set

FD\_ZERO(&readfds);

// Add the server socket to the set

FD\_SET(server\_fd, &readfds);

max\_sd = server\_fd;

// Add child sockets to the set

for (int i = 0; i < max\_clients; i++) {

sd = client\_socket[i];

// If valid socket descriptor, add to read list

if (sd > 0)

FD\_SET(sd, &readfds);

// Track the maximum socket descriptor

if (sd > max\_sd)

max\_sd = sd;

}

// Wait for an activity on one of the sockets, timeout is NULL (blocking)

activity = select(max\_sd + 1, &readfds, NULL, NULL, NULL);

if ((activity < 0) && (errno != EINTR)) {

printf("Select error\n");

}

// If something happened on the server socket, it's an incoming connection

if (FD\_ISSET(server\_fd, &readfds)) {

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

perror("Accept failed");

exit(EXIT\_FAILURE);

}

printf("New connection, socket fd: %d, ip: %s, port: %d\n",

new\_socket, inet\_ntoa(address.sin\_addr), ntohs(address.sin\_port));

// Send current date and time to the client

time\_t current\_time;

time(&current\_time);

char \*daytime = ctime(&current\_time);

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

// Add new socket to the array of client sockets

for (int i = 0; i < max\_clients; i++) {

if (client\_socket[i] == 0) {

client\_socket[i] = new\_socket;

printf("Added new client to list at index %d\n", i);

break;

}

}

}

// Else, it is an I/O operation on some other client socket

for (int i = 0; i < max\_clients; i++) {

sd = client\_socket[i];

if (FD\_ISSET(sd, &readfds)) {

// Check if it was for closing, and also read the incoming message

if ((valread = read(sd, buffer, BUFFER\_SIZE)) == 0) {

// Somebody disconnected, get their details and print

getpeername(sd, (struct sockaddr \*)&address, &addrlen);

printf("Host disconnected, ip: %s, port: %d\n",

inet\_ntoa(address.sin\_addr), ntohs(address.sin\_port));

// Close the socket and mark as 0 in the list for reuse

close(sd);

client\_socket[i] = 0;

} else {

// Echo the message back to the client (for illustration)

buffer[valread] = '\0';

send(sd, buffer, strlen(buffer), 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;

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

// 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;

}

// Receive the current time from the server

read(sock, buffer, BUFFER\_SIZE);

printf("Daytime from server: %s\n", buffer);

// Close the socket

close(sock);

return 0;

}