SERVER CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define MAXLINE 1024

int isValidIPv4(char *ip) {
    int num;
    int dots = 0;
    char *ptr;
    if (ip == NULL)
        return 0;
    ptr = strtok(ip, ".");
    if (ptr == NULL)
        return 0;
    while (ptr) {
        if (!strcmp(ptr, "")) // Check if part is empty
            return 0;
        num = atoi(ptr);
        if (num < 0 || num > 255)
            return 0;
        ptr = strtok(NULL, ".");
        if (ptr != NULL)
            dots++;
    }
    if (dots != 3)
    return 0;
    return 1;
}

char getClass(char *ip) {
    int first_octet;
    sscanf(ip, "%d", &first_octet);

    if (first_octet >= 0 && first_octet <= 127)
        return 'A';
    else if (first_octet >= 128 && first_octet <= 191)
        return 'B';
    else if (first_octet >= 192 && first_octet <= 223)
        return 'C';
    else if (first_octet >= 224 && first_octet <= 239)
        return 'D';
    else if (first_octet >= 240 && first_octet <= 255)
        return 'E';
    else
        return 'X'; // Invalid class
```

```
}

void printDetails(char *ip, char class, char *result) {
    int first_octet, second_octet, third_octet, fourth_octet;
    sscanf(ip, "%d.%d.%d.%d", &first_octet, &second_octet, &third_octet, &fourth_octet);

    unsigned long int network_id, broadcast_id, default_mask;

    switch (class) {
        case 'A':
            network_id = (first_octet << 24) | (0 << 16) | (0 << 8) | (0);
            broadcast_id = (first_octet << 24) | (255 << 16) | (255 << 8) | (255);
            default_mask = 0xFF000000;
            break;
        case 'B':
            network_id = (first_octet << 24) | (second_octet << 16) | (0 << 8) | (0);
            broadcast_id = (first_octet << 24) | (second_octet << 16) | (255 << 8) | (255);
            default_mask = 0xFFFF0000;
            break;
        case 'C':
            network_id = (first_octet << 24) | (second_octet << 16) | (third_octet << 8) | (0);
            broadcast_id = (first_octet << 24) | (second_octet << 16) | (third_octet << 8) | (255);
            default_mask = 0xFFFFFF00;
            break;
        default:
            sprintf(result, "Invalid IPv4 address class.\n");
            return;
    }

    sprintf(result, "Valid IPv4 address.\nClass: %c\nNetwork ID: %ld.%ld.%ld.%ld\nBroadcast
ID: %ld.%ld.%ld.%ld\nDefault Mask: %ld.%ld.%ld.%ld\n",
            class,
            (network_id >> 24) & 0xFF, (network_id >> 16) & 0xFF, (network_id >> 8) & 0xFF,
network_id & 0xFF,
            (broadcast_id >> 24) & 0xFF, (broadcast_id >> 16) & 0xFF, (broadcast_id >> 8) & 0xFF,
broadcast_id & 0xFF,
            (default_mask >> 24) & 0xFF, (default_mask >> 16) & 0xFF, (default_mask >> 8) & 0xFF,
default_mask & 0xFF);
}

int main() {
    int sockfd;
    char buffer[MAXLINE];
    struct sockaddr_in servaddr, cliaddr;

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
 memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
```

```
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);

    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    int len, n;
    len = sizeof(cliaddr);
    n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr *)&cliaddr,
&len);
    buffer[n] = '\0';

    char result[MAXLINE];
    if (isValidIPv4(buffer)) {
        char ipClass = getClass(buffer);
        printDetails(buffer, ipClass, result);
    } else {
        strcpy(result, "Invalid IPv4 address.\n");
    }
    sendto(sockfd, result, strlen(result), MSG_CONFIRM, (const struct sockaddr *)&cliaddr, len);

    printf("Details sent to the client.\n");

    close(sockfd);
    return 0;
}
```