בס"ד

חלק ב- traceroute

הסבר כללי על הקוד:
הקוד הזה משדר חבילות ICMP (Echo Request) לכל TTL, שואל שאלות על כל hop בדרך ליעד, ומחשב את זמני התגובה. לאחר שליחת שלוש חבילות לכל TTL, הוא סופר את התשובות ומפסיק לשלוח אם היעד הושג. במידה ולא הגענו ליעד אחרי 30 נתבים בדרך הוא יחזיר הודעות שגיאה

הסבר קוד-

הגדרת כמות הנתבים שניתן לעבור בדרך (ערך מקסימלי של TTL וגודל חבילה מקסימלי (בבתים)

```
#define MAX_TTL 30
#define PACKET_SIZE 64
```

הגדרת מבנה של חבילת icmp
המבנה מכיל את המבנה ICMPHDR
בו יש כותרת ה ICMP-שכוללת את סוג החבילה, קוד השגיאה, מזהה רצפים, וchecksum

```
struct icmp_packet {
    struct icmphdr hdr;       // ICMP header
    char data[PACKET_SIZE - sizeof(struct icmphdr)];
};
```

פונקציה זו מוודאת שהנתונים של החבילה לא השתנו בדרך ע"י חישוב סכום נתונים של החבילה (16 ביטים) עד שנשארים 16 ביטים אחרונים ומבצעים את ההפך של הסכום (אם אורך החבילה אי זוגי הבית האחרון ייבדק בנפרד)

```
// Function to calculate the checksum for ICMP and IP headers
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;

    // Loop over the buffer in 16-bit chunks and add them together
    for (sum = 0; len > 1; len -= 2)
        sum += *buf++;

    // If there's one byte left, add it to the sum
    if (len == 1)
        sum += *(unsigned char *)buf;

    // Add the overflow from the upper 16 bits to the lower 16 bits
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);

    // Return the one's complement of the sum
    result = ~sum;
    return result;
}
```

הפונקציה שולחת חבילות ICMP עם TTL עולה כדי לעקוב אחרי הדרך של החבילות אל היעד. היא יוצרת סוקט מסוג RAW (סוקט המאפשר שליחה וקבלה של חבילות ברמת הIP ללא תלות בפרוטוקול למעלה, אם יש שגיאה יוצאת) ושולחת חבילות ICMP Echo Request(חבילות פינג) כדי לבדוק את הנתיב ברשת.
הגדרת כתובת ה IP-של היעד שאליו אנחנו שולחים את ה Echo Requests-במקרה הזה זה הכתובת של השרת שאליו נרצה לבצע tracerout.

```
// Traceroute function that sends ICMP Echo Requests with increasing TTL
void traceroute(const char *destination) {
    int sockfd;                  // Socket file descriptor
    struct sockaddr_in dest_addr; // Destination address structure

    // Create a raw socket for ICMP protocol
    if ((sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)) < 0) {
        perror("socket creation failed");  // Error handling if socket creation fails
        exit(EXIT_FAILURE);
    }

    // Prepare the destination address
    memset(&dest_addr, 0, sizeof(dest_addr));
    dest_addr.sin_family = AF_INET;  // Set family to IPv4

    // Convert the destination IP address from text to binary form
    if (inet_pton(AF_INET, destination, &dest_addr.sin_addr) <= 0) {
        perror("Invalid destination address");  // Handle invalid address
        exit(EXIT_FAILURE);
    }
```

בס"ד

```
    // Loop over TTL values (1 to 30 hops)
    for (int ttl = 1; ttl <= MAX_TTL; ttl++) {
        setsockopt(sockfd, IPPROTO_IP, IP_TTL, &ttl, sizeof(ttl));  // Set TTL for the socket

        int received_replies = 0;  // Variable to count received replies
        char hop_ip[INET_ADDRSTRLEN] = {0};  // Buffer to store the IP of the hop
        printf("%2d ", ttl);  // Print TTL value (current hop number)
```

עוברים בלולאה מ1 עד הערך המקסימלי של נתבים בו חבילה יכולה לעבור (30)
setsockopt מאפשר להגדיר את ערך הTTL של הסוקט, כלומר את מספר ה-נתבים שמותר לחבילה לעבור בדרך עד שתפוג. אחרי כל TTL מאתחלים את משתנה received_replies שמספר את התשובות מהיעד ואת המערך hop_ip לשמירת כתובת IPשל כל נתב

```
// Send 3 ICMP packets for each TTL
for (int i = 0; i < 3; i++) {
    struct icmp_packet packet;
    memset(&packet, 0, sizeof(packet));  // Clear the packet structure

    // Set the ICMP Echo Request header fields
    packet.hdr.type = ICMP_ECHO;                // ICMP type (Echo Request)
    packet.hdr.un.echo.id = getpid();           // Set ID to process ID
    packet.hdr.un.echo.sequence = ttl * 3 + i; // Sequence number (unique per packet)

    // Calculate the checksum for the packet
    packet.hdr.checksum = checksum(&packet, sizeof(packet));
```

לכל נתב שעוברים דרכו שולחים 3 חבילות כל חבילה שנשלחת מקבלת מזהה ייחודי באמצעות ID ו Sequence number כדי שנוכל לעקוב אחרי החבילות ולוודא איזה מהן חזרה קודם. ומחשבים CHECKSUM עבור החבילה

```
if (sendto(sockfd, &packet, sizeof(packet), 0, (struct sockaddr *)&dest_addr, sizeof(dest_addr)) <= 0)
    perror("sendto failed");  // If sending fails, print an asterisk
    printf("* ");
    continue;
}

fd_set fds;  // Set up a file descriptor set to wait for a reply
struct timeval timeout;  // Timeout for select function
FD_ZERO(&fds);           // Clear the file descriptor set
FD_SET(sockfd, &fds);    // Add the socket to the set
timeout.tv_sec = 1;      // Set timeout to 1 second
timeout.tv_usec = 0;

int ready = select(sockfd + 1, &fds, NULL, NULL, &timeout);  // Wait for a reply
```

שליחת החבילה וקליטת תשובה
שומרים את זמן שליחת החבילה ושולחים אותה ליעד
מחכים למשך שניה אם לא הצלחנו (יכול להיות גם מטעמי בטיחות) מדפיסים *

```
if (ready > 0) {
    char buffer[1024];          // Buffer to store the incoming reply
    struct sockaddr_in reply_addr;  // Address of the reply sender
    socklen_t reply_len = sizeof(reply_addr);

    // If a reply is received, process it
    if (recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&reply_addr, &reply_len) > 0) {
        gettimeofday(&end, NULL);  // Record end time
        double elapsed_time = (end.tv_sec - start.tv_sec) * 1000.0;  // Calculate round trip time
        elapsed_time += (end.tv_usec - start.tv_usec) / 1000.0;

        // Extract the IP header from the reply buffer
        struct iphdr *ip_header = (struct iphdr *)buffer;
        struct icmphdr *icmp_header = (struct icmphdr *)(buffer + (ip_header->ihl * 4));

        inet_ntop(AF_INET, &reply_addr.sin_addr, hop_ip, sizeof(hop_ip));  // Convert IP address to str

        if (i == 0) {
            printf("%s ", hop_ip);  // Print IP address of the hop
        }

        printf("%.3fms ", elapsed_time);  // Print round trip time in milliseconds

        received_replies++;  // Increment the reply counter
```

במידה וקיבל תשובה, מחשב את הזמן שחלף מרגע שליחת החבילה ועד קבלת התשובה.
לאחר מכן, המרת הכתובת ה IP-של התשובה לאנגלית הדפסתה. בנוסף, מדפיסים את זמן המעבר.(RTT)

בס"ד

במקרה והתשובה הגיעה מהיעד, הפונקציה מודיעה על הגעה ליעד ומסיימת את התוכנית. אם לא התקבלה תשובה נדפיס כוכבית.

```
        // If the reply address matches the destination address and it's the 3rd reply, print succe
        if (reply_addr.sin_addr.s_addr == dest_addr.sin_addr.s_addr && i == 2) {
            printf("\nReached destination: %s\n", destination);
            close(sockfd);  // Close the socket after reaching the destination
            return;
        }
    }
} else {
    printf("* ");  // Print asterisks if no reply is received within the timeout
}
```

אחרי כל TTL הדפסת שורה חדשה. אם הגענו לסוף ה־ TTL ולא הצלחנו להגיע ליעד, מדפיסים הודעה על היעד הבלתי נגיש.

```
        printf("\n");  // Print a new line after each TTL
    }

    close(sockfd);  // Close the socket if the destination is unreachable
    printf("Destination unreachable.\n");  // Print an error message if destination is unreachable
}
```

המיין בודק אם המשתמש סיפק את הקלט הנכון (פרמטרים כמו a- וכתובת ipa) אם הכל תקין, מפעילה את הפונקציה traceroute עם כתובת ה־ IP שנמסרה.

```
int main(int argc, char *argv[]) {
    // Check if the user has provided the correct command line arguments
    if (argc != 3 || strcmp(argv[1], "-a") != 0) {
        fprintf(stderr, "Usage: %s -a <destination>\n", argv[0]);
        return 1;  // Exit with error if incorrect arguments
    }

    // Call the traceroute function with the given destination
    traceroute(argv[2]);
    return 0;
}
```

בס"ד