אוניברסיטת אריאל בשומרון, המחלקה למדעי המחשב
רשתות תקשורת, סמסטר ב' תשפ"ג

PING (ICMP & TCP)

ליאור וינמן 213081763
יועד תמר 213451818

5 בינואר 2023

תוכן עניינים

1 פרק א' - תיאור המערכת . . . 2
1.1 הרצת הפרויקט . . . 2
1.2 הקובץ היוצר - $Makefile$ . . . 3
1.3 מימוש פינג - $ping.c$ . . . 4
1.4 מימוש פינג משוכלל - $new\_ping.c$ . . . 7
1.5 מימוש טיימר - $watchdog.c$ . . . 9
2 פרק ב' - תעבורה ופלט . . . 12
2.1 פינג . . . 12
2.1.1 תעבורת $ICMP$ . . . 12
2.1.2 פלט התוכנית . . . 13
2.2 פינג משוכלל ללא עצירה . . . 14
2.2.1 תעבורת $ICMP$ . . . 14
2.2.2 תעבורת TCP . . . 15
2.2.3 פלט התוכנית . . . 17
2.3 פינג משוכלל עם עצירה . . . 18
2.3.1 תעבורת $ICMP$ . . . 18
2.3.2 תעבורת TCP . . . 19
2.3.3 פלט התוכנית . . . 20
3 ביבליוגרפיה . . . 21

1 פרק א' - תיאור המערכת

1.1 הרצת הפרויקט

הקבצים הניתנים להרצה הם $ping.c$ ו־$new\_ping.c$ בלבד. (שאר הקבצים הם קבצי עזר).
על מנת להריץ את הפרויקט - עלינו קודם כל להוריד את כל ארבעת הקבצים לתוך תיקיה משותפת על המחשב (יש להשתמש במערכת הפעלה $Linux\ Ubuntu\ LTS\ 22.04$ בלבד).
לאחר שהורדנו את הקבצים, נפתח תרמינל על התקיה הנתונה עם הקבצים ונכתוב את הפקודה "$make\ all$" הדבר יגרום ליצירת קבצי ההרצה של התוכניות הניתנות להרצה. (לאחר מכן מומלץ להריץ את הפקודה "$clear$" כדי לנקות את הפלט הסטנדרטי מהטרמינל).
כעת, אם ברצוננו להריץ את $ping.c$ - נכתוב בטרמינל את הפקודה "$sudo\ ./PartA < ip >$" (כאשר במקום $< ip >$ יש לרשום כתובת אליה אנחנו מעוניינים לשלוח את הבקשות). אם אנו מעוניינים ואם ברצוננו להריץ את $new\_ping.c$ נכתוב בטרמינל את הפקודה "$sudo\ ./PartB < ip >$".
לאחר שכתבנו את אחת מהפקודות שמלעיל, התוכנית פשוט תתחיל לרוץ (כלומר ישר תתחיל לשלוח ולקבל תעבורת פינג).

1.2 הקובץ היוצר - $Makefile$

הקובץ הנ"ל הוא הקובץ שממיר את קבצי הקוד (הכתובים בשפת C) ומייצר מהם קבצי הרצה על מנת שנוכל להריץ בפועל את התוכניות שאנו כותבים.
נעבור על הקוד:

```
1  CC = gcc
2  FLAGS = -Wall -g
3
4  .PHONY: all clean
5
6  all: ping new_ping watchdog
7
8  ping: ping.o
9          $(CC) $(FLAGS) -o PartA ping.o
10
11 new_ping: new_ping.o
12         $(CC) $(FLAGS) -o PartB new_ping.o
13
14 watchdog: watchdog.o
15         $(CC) $(FLAGS) -o watchdog watchdog.o
16
17 ping.o: ping.c
18         $(CC) $(FLAGS) -c ping.c
19
20 new_ping.o: new_ping.c
21         $(CC) $(FLAGS) -c new_ping.c
22
23 watchdog.o: watchdog.c
24         $(CC) $(FLAGS) -c watchdog.c
25
26 clean:
27         rm -f *.o PartA PartB watchdog
```

בקובץ זה ראשית אנו מייצרים מקבצי הקוד קבצי אובייקטים בינאריים (שורות $17-24$) תהליך זה הוא תהליך ההידור. לאחר מכן מהקבצים הבינאריים אנו מייצרים קבצי הרצה שבעזרתם אנו למעשה מריצים את הפרוייקט (שורות $8-15$) תהליך זה הוא תהליך הלינקוג'.

1.3 מימוש פינג - $ping.c$

נעבור על הקובץ $ping.c$:
בכל חבילה הנשלחת יש תא המיועד לביטים אשר מאמתים האם החבילה הגיעה בשלמותה ביטים אלו נקראים $checksum$. כאן כתבנו פונקציה לחישוב ולהשמה של ה־$checksum$:

```
unsigned short checksum(void *b, int len)
{
    unsigned short *buf = b;
        unsigned int sum=0;
        unsigned short result = 0;

        for ( sum = 0; len > 1; len -= 2 )
                sum += *buf++;
        if ( len == 1 )
                sum += *(unsigned char*)buf;

        sum = (sum >> 16) + (sum & 0xFFFF);
        sum += (sum >> 16);
        result = ~sum;
        return result;
}
```

כאן וידאנו שהמשתמש אכן הריץ את התוכנית כנדרש, ז"א נתן לתוכנית כתובת IP אליה לבצע $ping$:

```
    if (argc != 2) // checking that the user has specified an IP address
    {
        printf("usage: ./partb <ip>\n");
        exit(EXIT_FAILURE);
    }
```

כאן סידרנו את החבילה שאנו מעוניינים לשלוח, אנו שולחים "...000" (אין סיבה מיוחדת לדבר, סתם חבילה דיפולטיבית):

```
    for (size_t i = 0; i < MSG_LEN - 1; i++) //the message we sent
    {
        data[i] = '0';
    }

    data[MSG_LEN - 1] = '\0';
```

כאן פתחנו שקע שדרכו תעבור תעבורת פינג (*raw socket*):

```
    sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP); // creating an RAW socket for ICMP communication
    if (sock <= 0) // checking if socket created
    {
        perror("socket() failed");
        close(sock);
        exit(errno);
    }

    addr_ping.sin_family = AF_INET; // setting up the struct for ICMP communicationg
    if(inet_aton(argv[1], &addr_ping.sin_addr) < 0)
    {
        perror("inet_aton() failed");
        exit(errno);
    }
```

כאן אנו מסדרים את ההגדרות של החבילות הנשלחות:

```
        memset(&icmp, 0, sizeof(icmp)); // setting up the struct and packet for ICMP communication
        memset(&packet , 0 , sizeof(packet));
        icmp.type = ICMP_ECHO;
        icmp.un.echo.sequence = seq;
        icmp.checksum = 0;

        memcpy((packet), &icmp, ICMP_HDRLEN);
        memcpy(packet + ICMP_HDRLEN, data, data_len);

        icmp.checksum = checksum(&packet, sizeof(packet));
        memcpy((packet), &icmp, ICMP_HDRLEN);
```

כאן אנו שולחים בקשת פינג לכתובת ולאחר מכן מקבלים תשובת פינג. בנוסף, אנו מחשבים את הזמנים של כל התהליך:

```
        gettimeofday(&start , NULL); // starting counting ping-time
        if (sendto(sock, &packet, ICMP_HDRLEN + data_len, 0, (struct sockaddr*)&addr_ping, sizeof(addr_ping)) < 0) // sending ICMP-ECHO-REQUEST
        {
            perror("sendto() failed");
            close(sock);
            exit(errno);
        }

        addr_len = sizeof(addr_ping); // receiving ICMP-ECHO-REPLEY
        bzero(buffer, IP_MAXPACKET);
        len = recvfrom(sock, buffer, buffer_len, 0, (struct sockaddr *)&addr_ping, &addr_len);
        if (len == -1)
        {
            perror("recvfrom() failed");
            close(sock);
            exit(errno);
        }

        gettimeofday(&end , NULL); // ending counting ping-time

        time = (end.tv_sec - start.tv_sec) * 1000.0 + (end.tv_usec - start.tv_usec) / 1000.0; // saving the time in mili-seconds
```

כאן אנו מקבלים את הכתובת ממנה החבילה נשלחה כתגובה ומדפיסים למשתמש, בנוסף מבצעים המתנה של שניה:

```
        ip = (struct iphdr*)buffer;

        printf("%d bytes has been recv from %s to " , len, inet_ntoa(*(struct in_addr*)&ip->saddr));
        printf("%s: icmp_seq=%d ttl=%d time=%.2f ms\n", inet_ntoa(*(struct in_addr*)&ip->daddr) , icmp.un.echo.sequence , ip->ttl, time);

        seq++;

        sleep(1);
```

לבסוף אנו סוגרים את השקע שפתחנו ויוצאים בהצלחה מהתוכנית.

1.4 מימוש פינג משוכלל - $new\_ping.c$

נעבור על הקובץ $new\_ping.c$:
קובץ זה הוא הרחבה לקובץ הקודם, לכן רוב הדברים זהים לחלוטין למה שכבר ראינו (אזי לא נפרט עליהם שוב). נציג רק את התוספות החדשות של הקוד:
כאן אנו פותחים שקע לתעבורת TCP (המתבצעת עם ה־$watchdog$ בפורט 3000) ומגדירים את האובייקט שיחזיק את הכתובת של התקשורת:

```
        sockfd = socket(AF_INET, SOCK_STREAM, 0); // creating socket for TCP communcation
        if(sock <= 0) // checking if socket created
        {
            perror("socket() failed");
            close(sock);
            close(sockfd);
            exit(errno);
        }

        memset(&addr_server, '\0', sizeof(addr_server)); // setting up the struct for TCP communication
        addr_server.sin_family = AF_INET;
        addr_server.sin_port = htons(PORT);
        addr_server.sin_addr.s_addr = inet_addr(IP);
```

כאן אנו מריצים את ה־$watchdog$ (מריצים כעוד תהליך - במקביל):

```
        pid = fork(); // starting the watchdog proccess
        if (pid == 0)
        {
            execvp(args[0], args);
        }
```

כאן אנו ממתינים שניה עד שה־$watchdog$ יהיה מוכן לקבלת תקשורת, לאחר מכן אנו מתחברים אליו ושולחים אליו את כתובת ה־IP שהמשתמש הזין:

```
            sleep(1); // delay until watchdog is setted up

            if(connect(sockfd, (struct sockaddr*)&addr_server, sizeof(addr_server)) < 0) // connecting to watchdog
            {
                perror("connect() failed");
                close(sock);
                close(sockfd);
                exit(errno);
            }

            if(send(sockfd , argv[1] , strlen(argv[1]) + 1 , 0) <= 0) // sending IP to watchdog
            {
                perror("send() failed");
                close(sock);
                close(sockfd);
                exit(errno);
            }
```

כאן אנו אמורים לקבל הודעת OK מה־$watchdog$ שזהו סימן שיש תקשורת:

```
165             if(recv(sockfd , buffer , BUFSIZ , 0) <= 0) // receiving an OK message from watchdog
166             {
167                 perror("recv() failed");
168                 close(sock);
169                 close(sockfd);
170                 exit(errno);
171             }
172             if(strcmp(buffer, OK) != 0) // checking that OK received
173             {
174                 printf("error occurred!");
175                 close(sock);
176                 close(sockfd);
177                 exit(EXIT_FAILURE);
178             }
```

כאן אנו שולחים בחזרה הודעת OK וכמו כן בודקים מה מצב התהליך שרץ ברקע:

```
193             if(send(sockfd, OK, strlen(OK) + 1 , 0) < 0) // sending OK message
194             {
195                 perror("send() failed");
196                 close(sock);
197                 close(sockfd);
198                 exit(errno);
199             }
200
201             wait(&status);
202             if(status == -1)
203             {
204                 printf("proccess status failed\n");
205                 close(sock);
206                 close(sockfd);
207                 exit(EXIT_FAILURE);
208             }
```

חוץ מהדברים שפירטנו מלעיל, שאר הקובץ זהה לחלוטין (שכן הוא רק הרחבה לקובץ הקודם), לכן לא פירטנו על שאר הדברים.

1.5 מימוש טיימר - $watchdog.c$

הקובץ הנ"ל הוא למעשה קובץ בדיקת תקינות כתובת IP שהמשתמש הזין. הבדיקה מתרחשת על ידי הפעלת טיימר של 10 שניות של המתנה לתשובה מהכתובת (אחרי ששלחנו פינג בקשה) ואם לא התקבלה תשובה, אזי התוכנית תיעצר.
נעבור על הקובץ $watchdog.c$:
כאן כתבנו פונקציה אשר בודקת האם עבר הזמן והאם צריך לעצור או לא:

```
void timer_callback()
{
  if (!received_echo_reply)
  {
    printf("server <%s> cannot be reached.\n" , buffer); // printing unreach message

    kill(0 , SIGKILL); // killing all proccesses and exiting
  }
}
```

כאן פתחנו את השקע שדרכו תעבור התעבורה (TCP):

```
    server_sock = socket(AF_INET, SOCK_STREAM, 0); // creating the listener socket
    if(server_sock <= 0) // checking if socket created
    {
        perror("socket() failed");
        close(server_sock);
        exit(errno);
    }
```

כאן אנו מאפשרים שימוש חוזר בפורט ובכתובת ומגדירים מבנה שיחזיק את הכתובת והפורט לתקשורת:

```
    if(setsockopt(server_sock, SOL_SOCKET, SO_REUSEADDR, &er, sizeof(er)) < 0) // checking if ip and por
    {
        perror("setsockopt() failed");
        close(server_sock);
        exit(errno);
    }

    server_addr.sin_family = AF_INET; // setting up the struct for TCP communication
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr(IP);
```

כאן אנו מקשרים את הכתובת והפורט לשקע שפתחנו ומאזינים לתקשורת נכנסת, אם יש כזו מאשרים:

```
    if(bind(server_sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) // binding socket with settings
    {
        perror("bind() failed");
        close(server_sock);
        exit(errno);
    }

    if(listen(server_sock, CONNECTIONS) < 0) //listen to incoming connections
    {
        perror("listen() failed");
        close(server_sock);
        exit(errno);
    }

    addr_size = sizeof(client_addr);
    client_sock = accept(server_sock, (struct sockaddr*)&client_addr, &addr_size); //accept a connection
    if(client_sock <= 0) // checking if accepted
    {
        perror("accept() failed");
        close(client_sock);
        close(server_sock);
        exit(errno);
    }
```

כאן אנו מקבלים מהמשתמש את כתובת ה־IP ושולחים הודעת OK:

```
    if(recv(client_sock , buffer , BUFSIZ , 0) < 0) // receiving IP
    {
        perror("recv() failed");
        close(client_sock);
        close(server_sock);
        exit(errno);
    }

    if(send(client_sock, OK , strlen(OK) + 1 , 0) < 0) // sending OK message
    {
        perror("send() failed");
        close(client_sock);
        close(server_sock);
        exit(errno);
    }
```

כאן אנו מפעילים טיימר ל־10 שניות, לאחר מכן בודקים האם יש צורך לעצור את התוכנית:

```
    timer.it_value.tv_sec = 10; // setting up timer for 10 seconds
    timer.it_value.tv_usec = 0;
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    printf("waiting for ping response...\n");

    setitimer(ITIMER_REAL, &timer, NULL); // running the timer

    signal(SIGALRM, timer_callback);
```

כמובן שאם כן, אנו חוזרים לפונקציה למעלה שם מדפיסים הודעת שגיאה ויוצאים. אבל אם אין צורך, אז אנו מקבלים הודעת OK ויוצאים מהתוכנית בהצלחה:

```
    if(recv(client_sock , buff , BUFSIZ , 0) < 0) // receiving OK message
    {
        perror("recv() failed");
        close(client_sock);
        close(server_sock);
        exit(errno);
    }
    if(strcmp(buff, OK) != 0) // checking that OK received
    {
        printf("error occurred!");
        close(client_sock);
        close(server_sock);
        exit(EXIT_FAILURE);
    }


    close(client_sock);
    close(server_sock);
    exit(EXIT_SUCCESS);
```

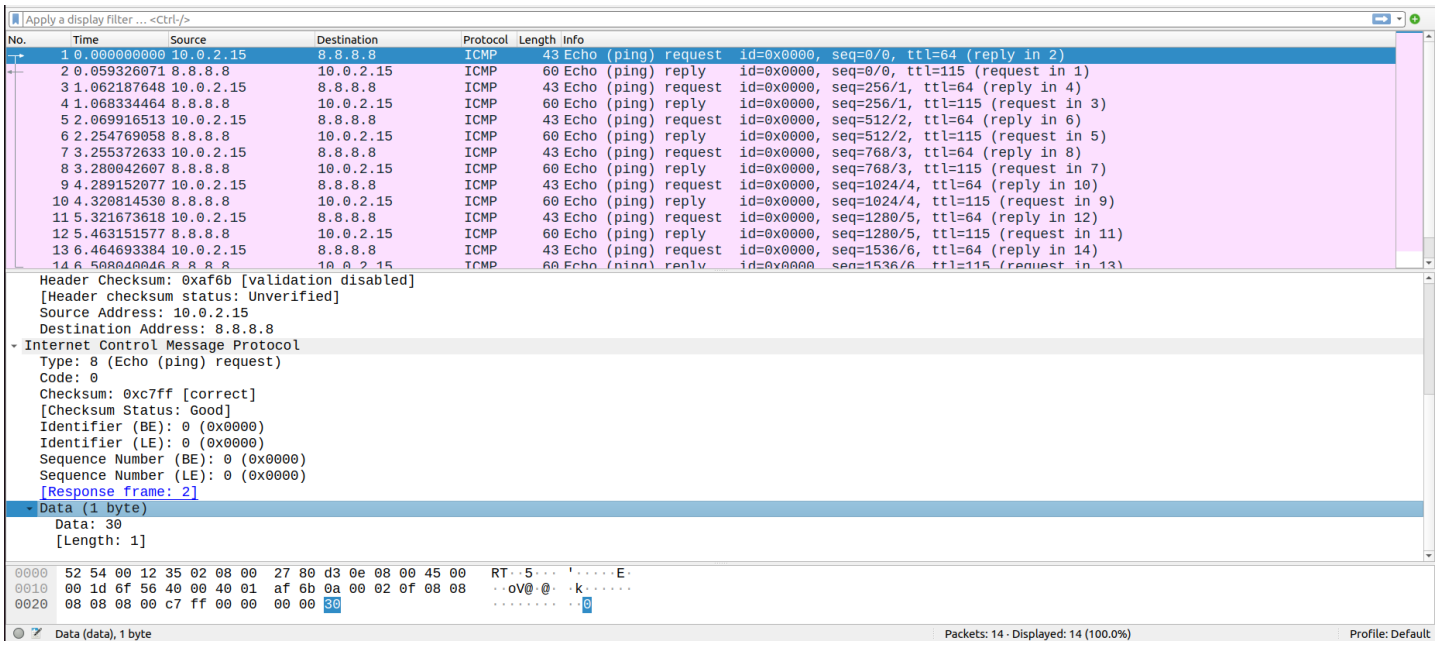
2 פרק ב' - תעבורה ופלט

2.1 פינג

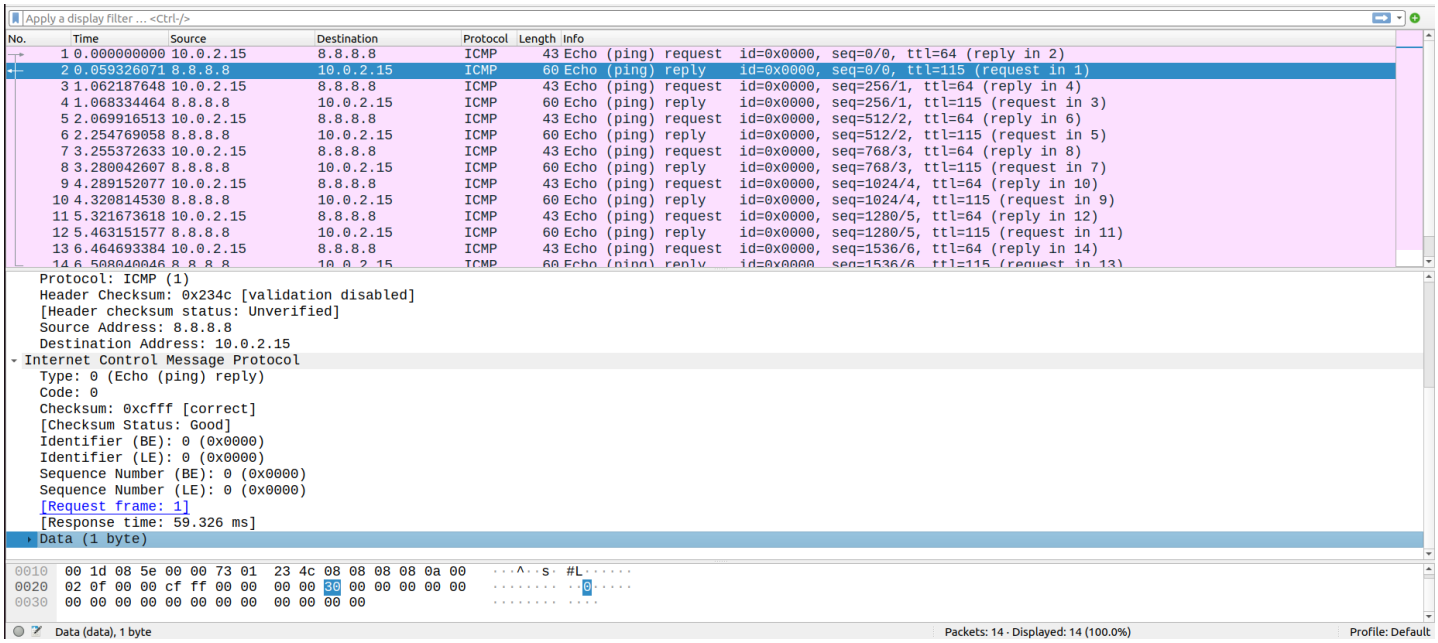
כאן אנו רואים את התעבורה והפלט של "$ping.c$".

2.1.1 תעבורת $ICMP$

כאן אנו יכולים לראות את התעבורה של הפינג, שליחה סטנדרטית של חבילות בקשה ולאחר מכן קבלה של חבילות תגובה.
חבילת בקשה:



חבילת תגובה:



2.1.2 פלט התוכנית

פלט התוכנית המתקבל הוא פלט סטנדרטי כמו של התוכנית $ping$ הסטנרטית של לינוקס.
כאן הרצנו את התוכנית 7 פעמים:

```
yoad@yoad-VirtualBox:~/Desktop/Ex4_cn/cnc_assignment4-1$ sudo ./PartA 8.8.8.8
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=0 ttl=115 time=59.44 ms
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=1 ttl=115 time=6.28 ms
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=2 ttl=115 time=185.08 ms
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=3 ttl=115 time=24.80 ms
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=4 ttl=115 time=31.81 ms
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=5 ttl=115 time=141.57 ms
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=6 ttl=115 time=43.48 ms
^C
yoad@yoad-VirtualBox:~/Desktop/Ex4_cn/cnc_assignment4-1$
```

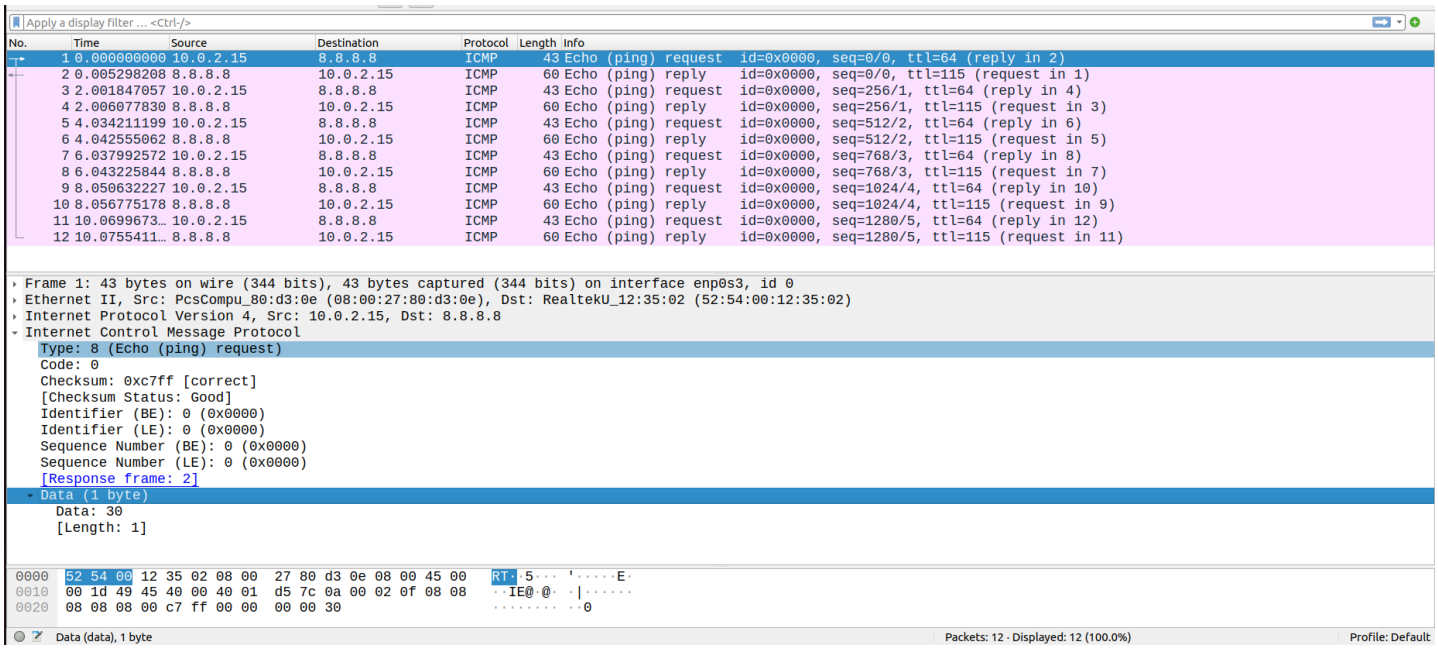
כמו כן, נשים לב כי בתעבורה שמלעיל יש בדיוק ($= 7 \cdot 2$)14 חבילות, שכן בכל ריצה של התוכנית נשלחת חבילת בקשה ומתקבלת חבילת תגובה כך שהפלט הגיוני מאד.

2.2 פינג משוכלל ללא עצירה

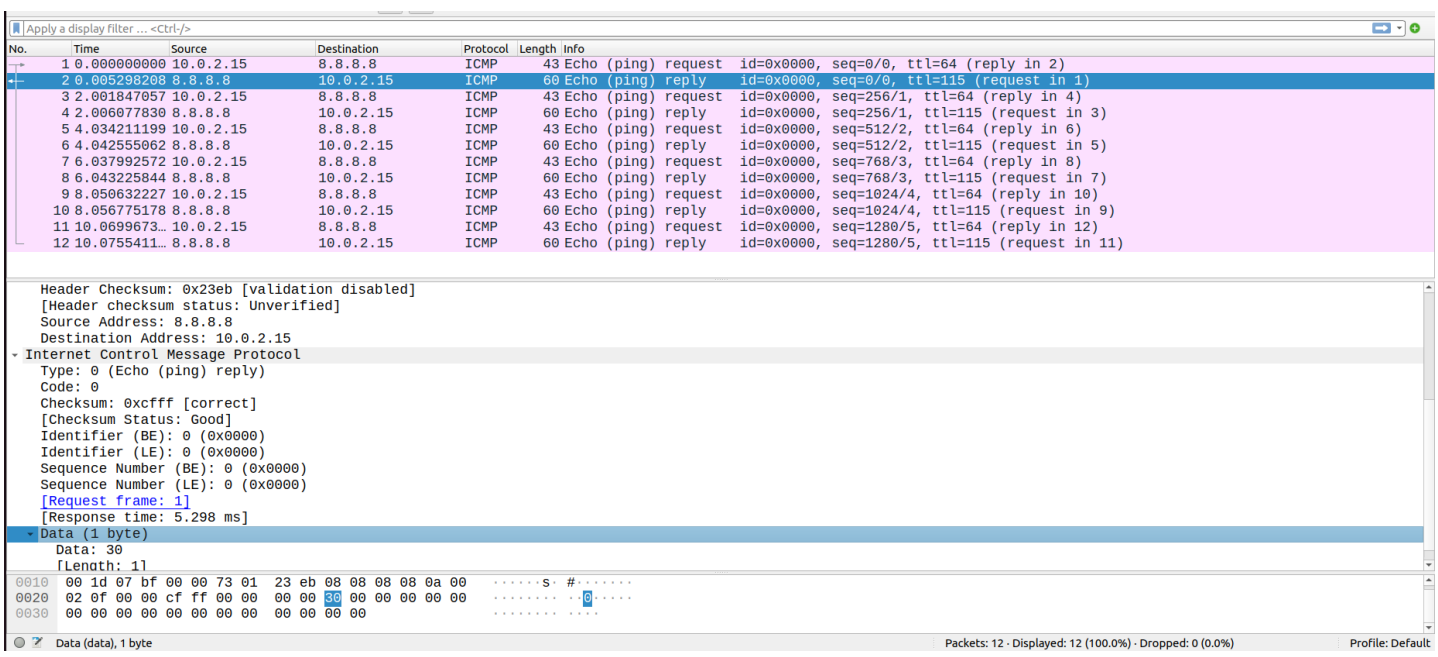
כאן אנו רואים את התעבורה והפלט של "*better_ping.c*" ללא עצירת ה־*watchdog*.

2.2.1 תעבורת *ICMP*

כאן אנו רואים חבילה של בקשת פינג:



כאן אנו רואים חבילה של תגובת פינג:



2.2.2 תעבורת TCP

כעת בגלל שיש לנו $watchdog$ ישנה גם תעבורת TCP. נתבונן בה.
כאן אנו רואים פתיחת קשר:

tcp

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	49278 → 3000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1574796754 TSecr=0 WS=128
2	0.000019407	127.0.0.1	127.0.0.1	TCP	74	3000 → 49278 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1574796754 TSecr=1…
3	0.000030140	127.0.0.1	127.0.0.1	TCP	66	49278 → 3000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1574796754 TSecr=1574796754

כאו אנו רואים קבלה של כתובת ה־IP:

tcp

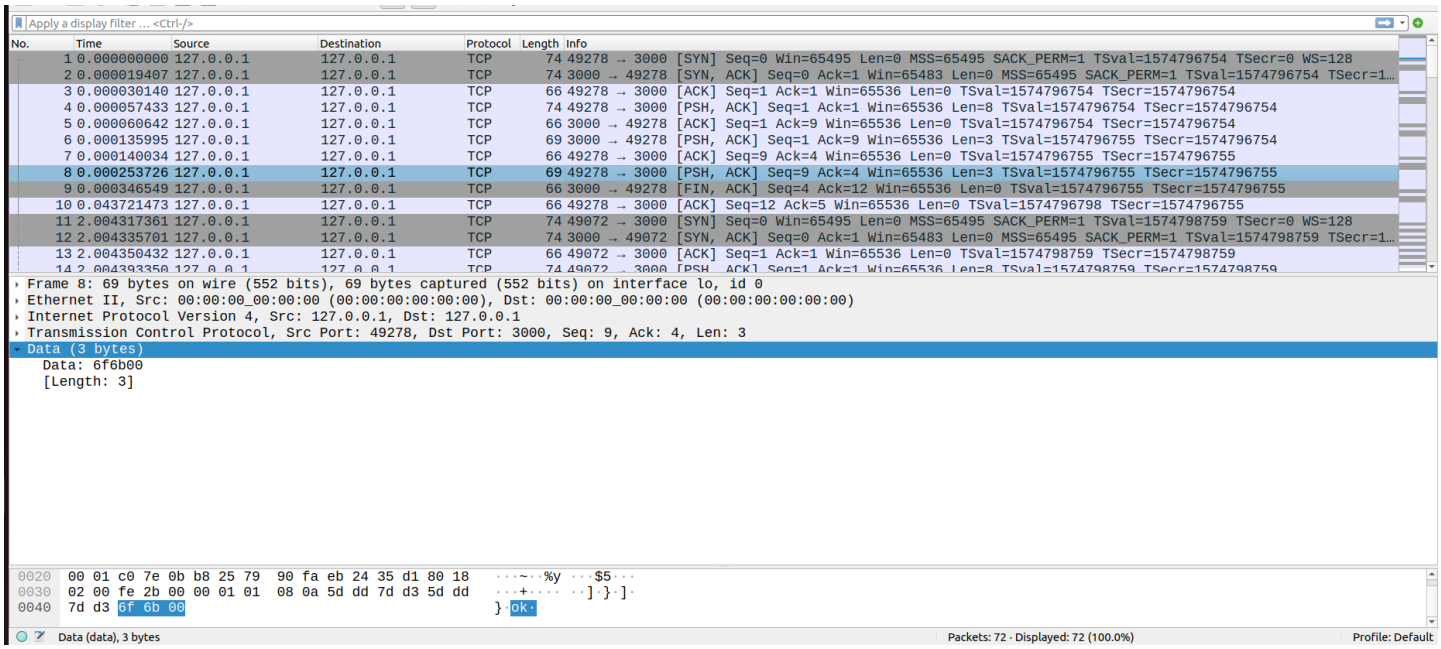
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	49278 → 3000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1574796754 TSecr=0 WS=128
2	0.000019407	127.0.0.1	127.0.0.1	TCP	74	3000 → 49278 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1574796754 TSecr=1…
3	0.000030140	127.0.0.1	127.0.0.1	TCP	66	49278 → 3000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1574796754 TSecr=1574796754
4	0.000057433	127.0.0.1	127.0.0.1	TCP	74	49278 → 3000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=8 TSval=1574796754 TSecr=1574796754
5	0.000060642	127.0.0.1	127.0.0.1	TCP	66	3000 → 49278 [ACK] Seq=1 Ack=9 Win=65536 Len=0 TSval=1574796754 TSecr=1574796754
6	0.000135995	127.0.0.1	127.0.0.1	TCP	69	3000 → 49278 [PSH, ACK] Seq=1 Ack=9 Win=65536 Len=3 TSval=1574796755 TSecr=1574796754
7	0.000140034	127.0.0.1	127.0.0.1	TCP	66	49278 → 3000 [ACK] Seq=9 Ack=4 Win=65536 Len=0 TSval=1574796755 TSecr=1574796755
8	0.000253726	127.0.0.1	127.0.0.1	TCP	69	49278 → 3000 [PSH, ACK] Seq=9 Ack=4 Win=65536 Len=3 TSval=1574796755 TSecr=1574796755
9	0.000346549	127.0.0.1	127.0.0.1	TCP	66	3000 → 49278 [FIN, ACK] Seq=4 Ack=12 Win=65536 Len=0 TSval=1574796755 TSecr=1574796755
10	0.043721473	127.0.0.1	127.0.0.1	TCP	66	49278 → 3000 [ACK] Seq=12 Ack=5 Win=65536 Len=0 TSval=1574796798 TSecr=1574796755
11	2.004317361	127.0.0.1	127.0.0.1	TCP	74	49072 → 3000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1574798759 TSecr=0 WS=128
12	2.004335701	127.0.0.1	127.0.0.1	TCP	74	3000 → 49072 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1574798759 TSecr=1…
13	2.004350432	127.0.0.1	127.0.0.1	TCP	66	49072 → 3000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1574798759 TSecr=1574798759
14	2.004393350	127.0.0.1	127.0.0.1	TCP	74	49072 → 3000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=8 TSval=1574798759 TSecr=1574798759

```
Acknowledgment Number: 1    (relative ack number)
Acknowledgment number (raw): 3945018830
1000 .... = Header Length: 32 bytes (8)
▸ Flags: 0x018 (PSH, ACK)
Window: 512
[Calculated window size: 65536]
[Window size scaling factor: 128]
Checksum: 0xfe30 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
▸ [Timestamps]
▸ [SEQ/ACK analysis]
TCP payload (8 bytes)
▾ Data (8 bytes)
Data: 382e382e382e3800
[Length: 8]
```

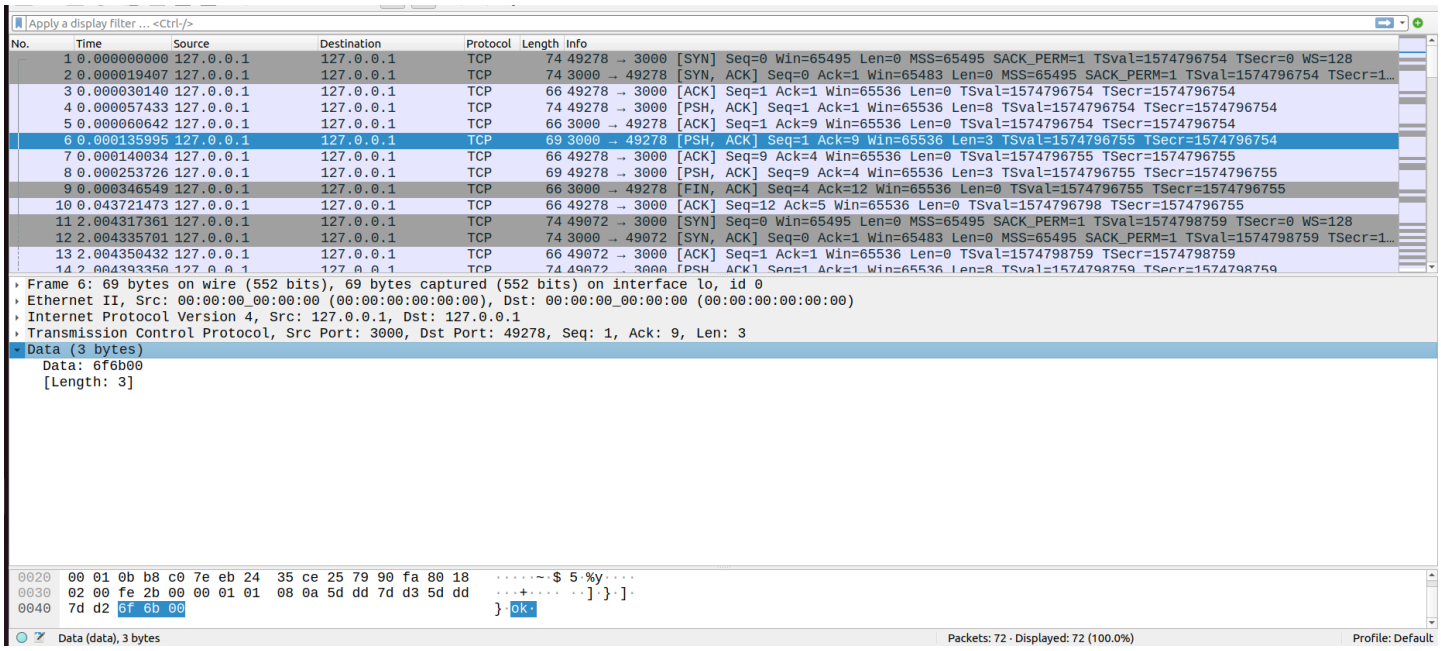
```
0020  00 01 c0 7e 0b b8 25 79  90 f2 eb 24 35 ce 80 18   ···~··%y ···$5···
0030  02 00 fe 30 00 00 01 01  08 0a 5d dd 7d d2 5d dd   ···0···· ··]·}·]·
0040  7d d2 38 2e 38 2e 38 2e  38 00                      }·8.8.8. 8·
```

Data (data), 8 bytes Packets: 72 · Displayed: 72 (100.0%) · Dropped: 0 (0.0%) Profile: Default

כאן אנו רואים שיש שליחה של הודעת OK:



כאן אנו רואים שיש קבלה של הודעת OK:



כאן אנו רואים סגירת קשר:

9	0.000346549	127.0.0.1	127.0.0.1	TCP	66	3000 → 49278 [FIN, ACK] Seq=4 Ack=12 Win=65536 Len=0 TSval=1574796755 TSecr=1574796755
10	0.043721473	127.0.0.1	127.0.0.1	TCP	66	49278 → 3000 [ACK] Seq=12 Ack=5 Win=65536 Len=0 TSval=1574796798 TSecr=1574796755

2.2.3 פלט התוכנית

פלט התוכנית שהתקבל:

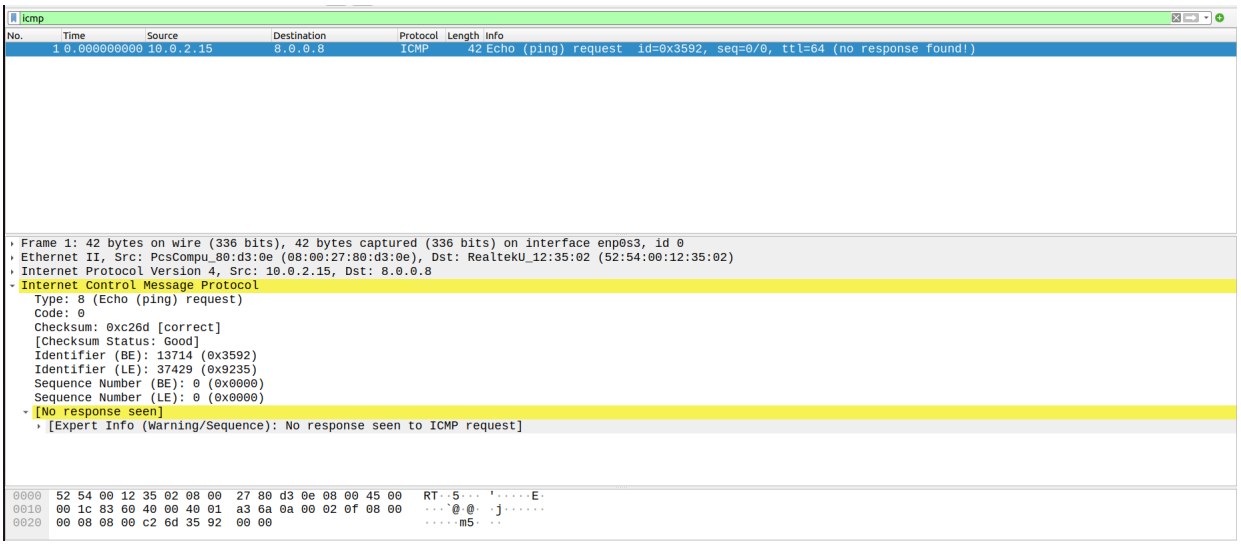
```
yoad@yoad-VirtualBox:~/Desktop/Ex4_cn/cnc_assignment4-1$ sudo ./PartB 8.8.8.8
waiting for ping response...
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=0 ttl=115 time=0.81 ms
waiting for ping response...
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=1 ttl=115 time=24.41 ms
waiting for ping response...
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=2 ttl=115 time=2.44 ms
waiting for ping response...
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=3 ttl=115 time=5.97 ms
waiting for ping response...
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=4 ttl=115 time=0.87 ms
waiting for ping response...
29 bytes has been recv from 8.8.8.8 to 10.0.2.15: icmp_seq=5 ttl=115 time=0.62 ms
^C
yoad@yoad-VirtualBox:~/Desktop/Ex4_cn/cnc_assignment4-1$
```

2.3 פינג משוכלל עם עצירה

כאן אנו רואים את התעבורה והפלט של "*better_ping.c*" עם עצירת ה־*watchdog*.

2.3.1 תעבורת *ICMP*

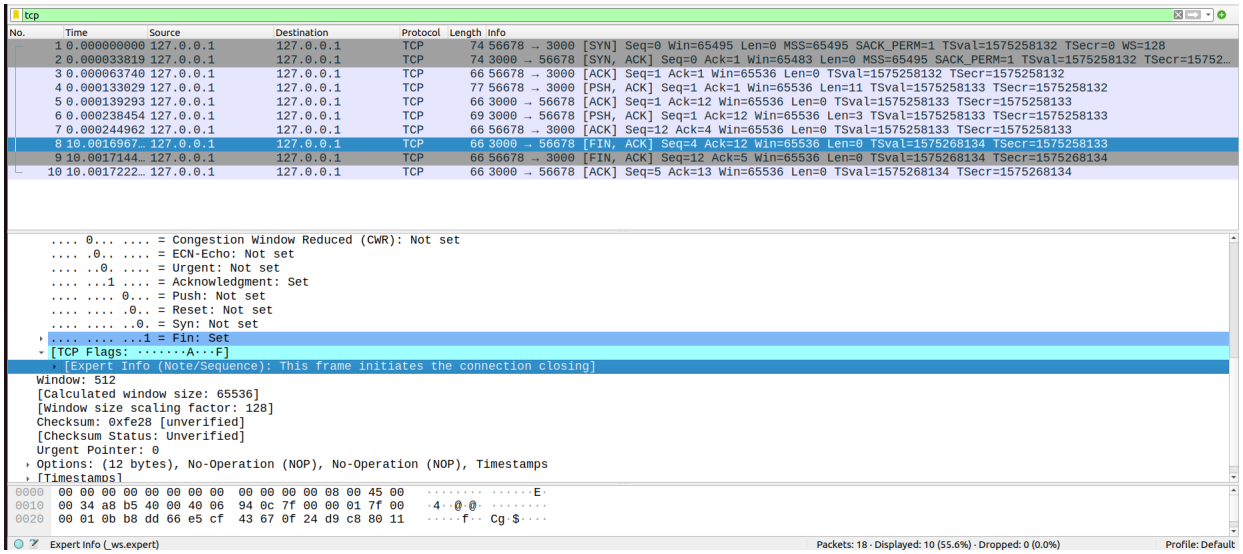
כאן אנו רואים חבילה בודדת של בקשת פינג, נשים לב שאין תגובה:



זו כל התעבורה כיוון שלאחר החבילה הזו ה־*watchdog* עצר את התוכנית.

2.3.2 תעבורת TCP

כאן אנו רואים יצירת קשר וסגירת קשר אחת ורואים את כלל התהליכים שכבר ראינו קודם לכן:



2.3.3 פלט התוכנית

פלט התוכנית שהתקבל:

```
yoad@yoad-VirtualBox:~/Desktop/Ex4_cn/cnc_assignment4-1$ sudo ./PartB 101.5.40.1
waiting for ping response...
server <101.5.40.1> cannot be reached.
Killed
yoad@yoad-VirtualBox:~/Desktop/Ex4_cn/cnc_assignment4-1$
```

3 ביבליוגרפיה

רשימת מקורות בהם השתמשנו בעת מימוש וכתיבת הפרוייקט:

1. $https://www.geeksforgeeks.org/fork-system-call/$
2. $https://www.digitalocean.com/community/tutorials/execvp-function-c-plus-plus$
3. $https://www.techtarget.com/searchnetworking/definition/ICMP$
4. מצגות הקורס "תכנות מערכות 1" של ד"ר חוגי אסף.
5. מצגות וחוברת הקורס "רשתות תקשורת".