```
struct ethhdr
{
unsigned char   h_dest[ETH_ALEN];       /* destination eth addr */
unsigned char   h_source[ETH_ALEN];     /* source ether addr    */
unsigned short h_proto;                 /* packet type ID field  */
};



struct arphdr
{
unsigned short ar_hrd;          /* format of hardware address   */
unsigned short ar_pro;          /* format of protocol address   */
unsigned char   ar_hln;         /* length of hardware address   */
unsigned char   ar_pln;         /* length of protocol address   */
unsigned short ar_op;           /* ARP opcode (command)         */
};


struct arp
{
        struct ethhdr    ethhdr;
        struct arphdr    arphdr;
        unsigned char   ar_sha[ETH_ALEN];  /* sender hardware address      */
        unsigned char   ar_sip[IP_ALEN];   /* sender IP address        */
        unsigned char   ar_tha[ETH_ALEN];  /* target hardware address*/
        unsigned char   ar_tip[IP_ALEN];   /* target IP address        */
};
```

```
//      Plik własny my_ping.h

#include   <netinet/ip.h>
#include   <linux/if_arp.h>
#include   <signal.h>
#include   <unistd.h>
#include   <stdio.h>
#include   <sys/ioctl.h>

struct arp
{
. . .
};
// definicje symboli

// zmienne globalne

unsigned char   *nazwa_interfejsu;    /* pobrana jako parametr programu */
struct ifreq       *opis_interfejsu;
in_addr_t  ip_interfejsu;                  /* w postaci sieciowej */
unsigned char   mac_interfejsu[ETH_ALEN];
in_addr_t  ip_odbiorcy;                    /* w postaci sieciowej */

struct ifconf      ifconf;
unsigned char   ioctl_buf[BUFSIZE];

unsigned char    bufor_odbiorczy[BUFSIZE];
unsigned char    bufor_nadawczy[BUFSIZE];
struct sockaddr_ll  adres_nadawczy;
struct sockaddr_ll  adres_odbiorczy;

int      gniazdo;
int     jest_pakiet = 0;

// prototypy funkcji
        . . .
```

```
//      plik główny my_arp.c


int main(int argc, char **argv)
{

        struct ifreq            *ifr;
        struct sockaddr_in      *sa;
        struct sockaddr         *ha;
        unsigned char           *ptr;
        int i;

        nazwa_interfejsu = argv[1];
        ip_odbiorcy=inet_addr(argv[2]);

//              Umieszczenie w strukturze adresowej adresu serwera

        memset((char *) &adres_nadawczy, 0, sizeof(adres_nadawczy)) ;
        adres_nadawczy.sll_ifindex = if_nametoindex(nazwa_interfejsu);
        memset((char *) adres_nadawczy.sll_addr, 0xff, 8);

        if ((gniazdo = socket(PF_PACKET, SOCK_RAW,
                                htons(ETH_P_ARP))) < 0) {
              perror("BLAD UTWORZENIA GNIAZDA");
             _exit(1); }

//      ***********         POBRANIE PARAMETROW GNIAZDA

        ifconf.ifc_len = BUFSIZE;
        ifconf.ifc_buf = (char *)ioctl_buf;
        if (ioctl(gniazdo, SIOCGIFCONF, &ifconf) < 0) {
              perror("BLAD IOCTL1");
             _exit(1); }
```

```
//      ***********         ODSZUKANIE OPISU INTERFEJSU NADAWCY

        opis_interfejsu = NULL;
        for (ptr = ioctl_buf; ptr < ioctl_buf + ifconf.ifc_len;) {
                ifr = (struct ifreq *)ptr;
                if (strcmp(ifr->ifr_name, nazwa_interfejsu) == 0 ) {
                        opis_interfejsu = ifr;
                        break;  }
                ptr += sizeof(struct ifreq);}
        if (opis_interfejsu == NULL ) {
                printf("Nie ma interfejsu %s\n", nazwa_interfejsu);
                exit(1); }
        printf("INTERFEJS %s:",opis_interfejsu->ifr_name);

//      ***********         POBRANIE ADRESU IP INTERFEJSU NADAWCY

        if (ioctl(gniazdo, SIOCGIFADDR, opis_interfejsu) < 0) {
                perror("BLAD IOCTL2");
               _exit(1); }
        sa = (struct sockaddr_in *)&(opis_interfejsu->ifr_addr);
        ip_interfejsu = sa->sin_addr.s_addr;
        printf("    IP: %s",inet_ntoa(ip_interfejsu));

//      ***********   POBRANIE ADRESU FIZYCZNEGO INTERFEJSU NADAWCY

        if (ioctl(gniazdo, SIOCGIFHWADDR, opis_interfejsu) < 0) {
                perror("BLAD IOCTL3");
               _exit(1); }

        ha = &(opis_interfejsu->ifr_hwaddr);
        ptr = ha->sa_data;
        memcpy(mac_interfejsu,ptr,ETH_ALEN);
        printf("    MAC: ");
        for (i=0; i<ETH_ALEN; i++) printf ("%x:",mac_interfejsu[i]);
        printf("\n");
```

```
        signal(SIGALRM, Obsluga_zegara);
        Nadaj_komunikat_ARP_REQUEST();

        alarm(10);
        Odbierz_komunikat();
        exit(0);
}


void Odbierz_komunikat(void)
{
        int     dl_adr;
        int     n;
        struct arp  *arp;

        dl_adr=sizeof(adres_odbiorczy);
        n = recvfrom(gniazdo, bufor_odbiorczy, BUFSIZE, 0,
              (struct sockaddr *) &adres_odbiorczy, &dl_adr);
        if (n < 0)    perror("BLAD RECVFROM");

        arp = (struct arp *) bufor_odbiorczy;
        if (arp->arphdr.ar_op == htons(ARPOP_REPLY))
              Obsluz_komunikat_ARP_REPLY(arp);
        else {
              printf("Odebrano pakiet: %d", ntohs(arp->arphdr.ar_op));
              return; }
}


void Obsluga_zegara(int nr_sygnalu)
{
        if (jest_pakiet) return;
        printf("Przekroczony czas oczekiwania (TIME OUT)\n");
        close(gniazdo);
        exit(1);
}
```

```
void Nadaj_komunikat_ARP_REQUEST(void)
{
        struct arp  *arp;
        int i;

        arp = (struct arp *) bufor_nadawczy;
        memset((char *) arp, 0, BUFSIZE) ;

        memset(arp->ethhdr.h_dest, 0xff, ETH_ALEN);
        memcpy(arp->ethhdr.h_source, mac_interfejsu,ETH_ALEN);
        arp->ethhdr.h_proto = htons(ETH_P_ARP);

        arp->arphdr.ar_hrd = htons(ARPHRD_ETHER);  /* 1 */
        arp->arphdr.ar_pro = htons(ETH_P_IP);      /* 0x0800 */
        arp->arphdr.ar_hln = ETH_ALEN;             /* 6 */
        arp->arphdr.ar_pln = IP_ALEN ;             /* 4 */
        arp->arphdr.ar_op = htons(ARPOP_REQUEST); /* 1 */

        memcpy(arp->ar_sip,(char *)&ip_interfejsu,IP_ALEN);
        memcpy(arp->ar_tip,(char *)&ip_odbiorcy,IP_ALEN);
        memcpy(arp->ar_sha, mac_interfejsu,ETH_ALEN);


        printf("Wyslano  ARP_REQUEST FROM %s(%x",
                        inet_ntoa(ip_interfejsu), mac_interfejsu[0]);
        for (i=1; i<ETH_ALEN; i++) printf (":%x",mac_interfejsu[i]);
        printf(")    TO: %s\n",inet_ntoa(ip_odbiorcy));

        if (sendto(gniazdo, bufor_nadawczy, sizeof(struct arp), 0,
                                (struct sockaddr *)&adres_nadawczy,
                                sizeof(adres_nadawczy))<0)
                perror("Blad w sendto");
}
```

```
void Obsluz_komunikat_ARP_REPLY(struct arp *arp)
{
        int     i;
        in_addr_t  ip_nadawcy_repl;
        in_addr_t  ip_odbiorcy_repl;

        memcpy((char *)&ip_nadawcy_repl, arp->ar_sip,IP_ALEN);
        memcpy((char *)&ip_odbiorcy_repl, arp->ar_tip,IP_ALEN);

        printf("Odebrano   ARP_REPLY FROM %s(%x",
                                inet_ntoa(ip_nadawcy_repl), arp->ar_sha[0]);
        for (i=1; i<ETH_ALEN; i++) printf (":%x",arp->ar_sha[i]);
        printf(")    TO: %s(%x",inet_ntoa(ip_odbiorcy_repl), arp->ar_tha[0]);
        for (i=1; i<ETH_ALEN; i++) printf (":%x",arp->ar_tha[i]);
        printf(")\n");
        return;
}
```