VI. ХАВСРАЛТ

**Хавсралт-1 IP хаягийн завсар оруулж байгаа хэрэгжүүлэлт**

for ( i = 0; i < 20 ; i++ )

{

a[i]=RANGE[i]; //"range"-n buh utgiig too helbereer "a"-ruu huulj avah;

}

for(i = 0; i < 20; i++)

{

if (a[i]==46) //hamgiin suuliin "."-temdegtiin index-g ni "coma"-ruu hadgalah;

{

++coma;

if(coma==3)

coma=i;

}

if (a[i]==45) //"-" temdegtees hoish 3 uragsh 3 temdegtiing "low" bolon "high" ruu huulj avah

{

low[0]=a[i-3];

low[1]=a[i-2];

low[2]=a[i-1];

high[0]=a[i+1];

high[1]=a[i+2];

high[2]=a[i+3];

}

}

if ( low [0] != 46 && low [1] !=46 && low [2] !=46 ) /\* "low" bolon "high"-g zadalj "l" bolon "h" int helbereer oruulah \*/

{

l=(low[2]-48)+(low[1]-48)\*10+(low[0]-48)\*100;

}

else if (low[0]==46 && low[1]!=46 && low[2]!=46 )

{

l=(low[1]-48)\*10+(low[2]-48);

}

else if (low[0]!=46 && low[1]==46 && low[2]!=46 )

{

l=low[2]-48;

}

if ( high [1] !=0 && high[2]!=0 )

{

h=(high[0]-48)\*100+(high[1]-48)\*10+high[2]-48;

}

else if ( high [1] !=0 && high[2]==0 )

{

h=(high[0]-48)\*10+high[1]-48;

}

else if ( high [1] == 0 && high[2]==0 )

{

h=high[0]-48;

}

if(l<0 || h<0 || l>255 || h>255){ //hervee ip hayagiin range buruu bval aldaa shideed gargana;

fprintf(stderr, "THIS IS NOT IP ADDRESS !!! \n");

quit(0);

}

RANGE[coma+4]=RANGE[coma+5]=RANGE[coma+6]=0;

for(l; l<=h; l++){ /\* RANGE-g "l"-ees "h" hurtel davtaj tuunii utgiig ni RANGE-d olgoh \*/

if (l>=100)

{

RANGE[coma+1]=l/100+48;

RANGE[coma+2]=(l-l/100\*100)/10+48;

RANGE[coma+3]=(l-l/100\*100-(l-l/100\*100)/10\*10)+48;

}

else if (l>=10 && l<100){

RANGE[coma+3]=0;

RANGE[coma+1]=l/10+48;

RANGE[coma+2]=l-(l/10)\*10+48;

}

else if(l>=1 && l<10){

RANGE[coma+3]=RANGE[coma+2]=0;

RANGE[coma+1]=l+48;

}

fprintf(stdout, "RANGE OF IP : %s \n",RANGE);

**Хавсралт-2 TCP флаг нэмэлтийн хэрэгжүүлэлт**

unsigned short in\_cksum(unsigned short \*ptr, int nbytes)

{

register long sum; /\* assumes long == 32 bits \*/

u\_short oddbyte;

register u\_short answer; /\* assumes u\_short == 16 bits \*/

/\*

\* Our algorithm is simple, using a 32-bit accumulator (sum),

\* we add sequential 16-bit words to it, and at the end, fold back

\* all the carry bits from the top 16 bits into the lower 16 bits.

\*/

sum = 0;

while (nbytes > 1) {

sum += \*ptr++;

nbytes -= 2;

}

/\* mop up an odd byte, if necessary \*/

if (nbytes == 1) {

oddbyte = 0; /\* make sure top half is zero \*/

\*((u\_char \*) &oddbyte) = \*(u\_char \*)ptr; /\* one byte only \*/

sum += oddbyte;

}

/\*

\* Add back carry outs from top 16 bits to low 16 bits.

\*/

sum = (sum >> 16) + (sum & 0xffff); /\* add high-16 to low-16 \*/

sum += (sum >> 16); /\* add carry \*/

answer = ~sum; /\* ones-complement, then truncate to 16 bits \*/

return(answer);

}

int knocker\_portscan\_by\_hostname (unsigned short port, unsigned int src\_addr, unsigned int dst\_addr, char \*mode, int speed)

{

int bla;

/\* for select() \*/

fd\_set readfs;

struct timeval tv;

/\* tcp header \*/

struct tcphdr send\_tcp;

/\* packet \*/

struct recv\_tcp

{

struct iphdr ip;

struct tcphdr tcp;

unsigned char data[65535];

} recv\_tcp;

/\* ip header \*/

struct pseudo\_header

{

unsigned int source\_address;

unsigned int dest\_address;

unsigned char placeholder;

unsigned char protocol;

unsigned short tcp\_length;

struct tcphdr tcp;

} pseudo\_header;

int sock;

struct sockaddr\_in sin;

static int add = 0;

add++;

send\_tcp.source = getpid() + add; /\* source port \*/

send\_tcp.dest = htons(port); /\* dest port \*/

send\_tcp.seq = getpid() + add; /\* seq number \*/

send\_tcp.ack\_seq = 0;

send\_tcp.res1 = 0;

send\_tcp.doff = 5;

send\_tcp.fin = 0;

send\_tcp.syn = 0;

send\_tcp.rst = 0;

send\_tcp.psh = 0;

send\_tcp.ack = 0; /\* ack number \*/

send\_tcp.urg = 0;

send\_tcp.window = htons(512); /\* window size \*/

send\_tcp.check = 0;

send\_tcp.urg\_ptr = 0;

if(!strcmp (mode, SYN)){

send\_tcp.syn = 1;

}

else if(!strcmp (mode, FIN)){

send\_tcp.fin = 1;

}

else if(!strcmp (mode, XMAS)){

send\_tcp.fin = 1;

send\_tcp.psh = 1;

send\_tcp.urg = 1;

}

else fprintf(stderr, "WRONG TYPE OF INPUT PLEASE TRY AGAIN ");

/\* ip header \*/

pseudo\_header.source\_address = src\_addr; /\* my ip \*/

pseudo\_header.dest\_address = dst\_addr; /\* dest ip \*/

pseudo\_header.placeholder = 0;

pseudo\_header.protocol = IPPROTO\_TCP; /\* tcp/ip \*/

pseudo\_header.tcp\_length = htons(20);

bcopy(&send\_tcp, &pseudo\_header.tcp, 20);

send\_tcp.check = in\_cksum((unsigned short \*)&pseudo\_header, 32);

/\* where is the socket? \*/

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(port);

sin.sin\_addr.s\_addr = dst\_addr;

sock = socket(AF\_INET, SOCK\_RAW, IPPROTO\_TCP);

if(sock < 0)

{

printf("couldnt open socket\n");

exit(1);

}

/\* set timer and prepare for select() \*/

tv.tv\_sec=0;

tv.tv\_usec=speed;

FD\_ZERO(&readfs);

FD\_SET(sock, &readfs);

/\* send paket \*/

sendto(sock, &send\_tcp, 20, 0, (struct sockaddr \*)&sin, sizeof(sin));

/\* "sniff" answer \*/

while(1)

{

bla=select(sock+1, &readfs, NULL, NULL, &tv);

if(bla)

read(sock, (struct recv\_tcp \*)&recv\_tcp, 65535); /\* new data \*/

else

return PORT\_IS\_OPEN; /\* port open \*/

if(recv\_tcp.tcp.dest == (getpid() + add))

{

/\* yes! \*/

close(sock);

if(recv\_tcp.tcp.rst == 1) {

return PORT\_IS\_CLOSED;

}

else

return PORT\_IS\_OPEN;

}

}

}