Buletinul Ştiinţific al Universităţii din Baia Mare
Seria B, Matematică-Informatică, vol.XII (1996),257-266

Dedicated to the 35th anniversary of the University of Baia Mare

AN IMAGE BROADCASTING PROTOCOL

Ovidiu COSMA

Objectives

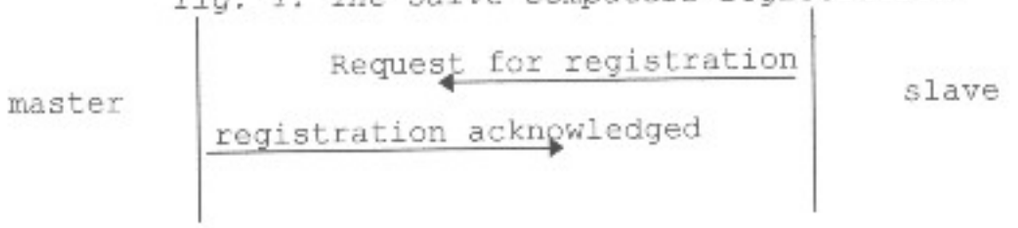
The aim of this paper is to present a communication protocol for local area networks, designed to improve the teaching process. The efficiency of laboratories increases, if the students can follow the explanations directly in front of the computers. But the problem is that maximum two persons have room close enough to the teacher to watch the screen. By using the protocol in this paper, this inconvenient can be solved. Thus the image from one of the computers (source), can be reproduced on the teacher's computer, or on all the computers in the classroom (destination). The source can be the teacher's computer, for presentations, and one of the students computer, for verifications. All the explanations can be performed directly on screen, and the blackboard is no more required.

Description

This protocol is based on a master - slave communication. The master is the teacher's computer and the slaves are all the other computers in the classroom. All the major actions are performed at master's commands.

The slaves activity begins with the request for registration (*fig.1*). After receiving the confirmation, they switch to normal operation, and the protocol dialogue continues in background.

fig. 1. The salve computers registration



There are three operation modes provided for the slaves:

1. *"Usual"*; Normal operation, without sending or receiving image packets;
2. *"Mirror"*; Continuously receiving and reproducing images, with the keyboard locked;
3. *"Source"*; Normal operation, but periodically sending the screen images in background.

All three operation modes are found again at the master computer, but with the difference that the keyboard is still active in the mirror mode, to get the operator's "back to usual" command. The master's operating modes can be switched with keyboard commands, that are converted into command packets to change the slaves operating modes also.

The next figures illustrate the master - slave dialogue.

fig. 2. Broadcasting master's screen images (source = master, destination = all slaves)

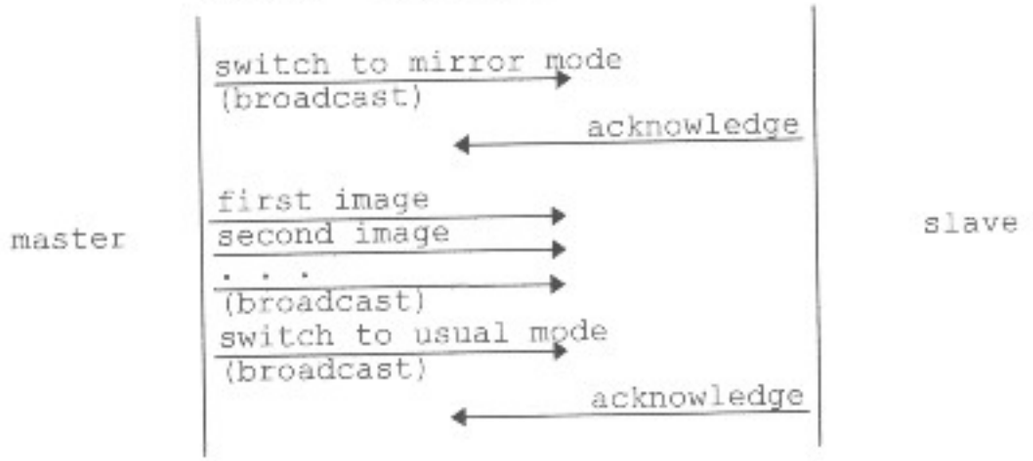


fig. 3. Broadcasting images from slaves (source=one salve, destination=master+all other slaves)

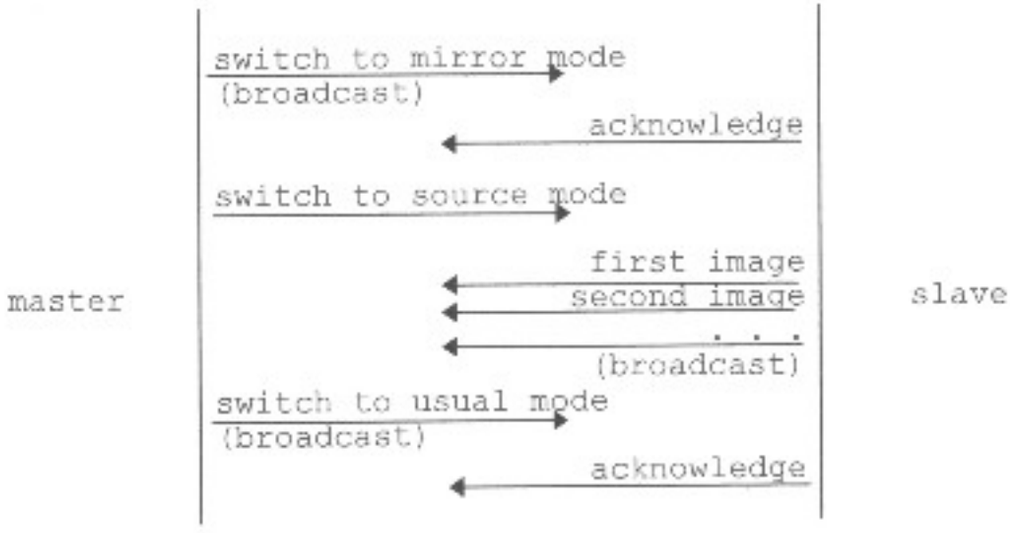
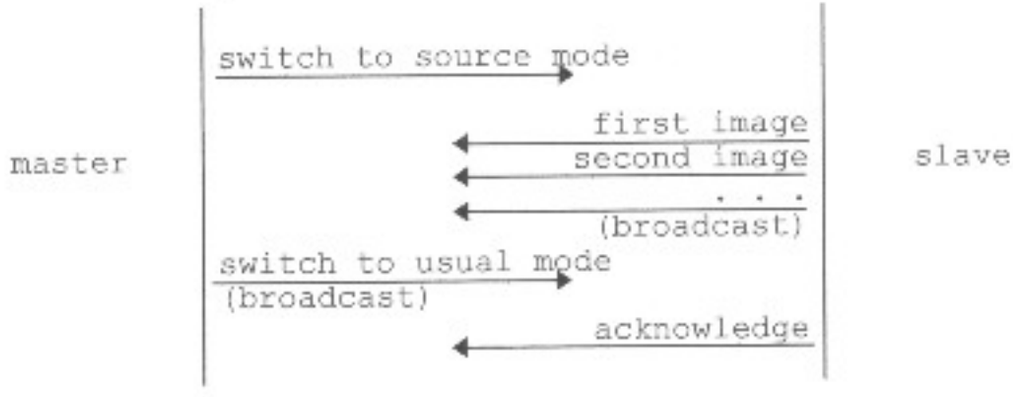


fig. 4. Transferring images from slaves (source = one salve, destination = master)



Implementation

The Image Broadcasting Protocol (IBP) is situated on top of the IPX protocol. IPX operates at the network level and takes care of the package exchange problems. The delivery and the order of packages is not guaranteed by the IPX, but in the case of screen images, the speed is much more important. The commands in the IBP are protected to communication errors, with acknowledgement messages from their destinations. Broadcasting of packages is possible with IPX, but within the same segment of network.

The IBP contains two types of modules, one for the master computer and the other for the slaves, both operating in

background. The IBP is controlled from the master's keyboard, with special combinations of keys. The slaves get all their commands from the communication media. The communication core of the IBP uses two IPX sockets. One of the sockets is reserved for images and the other for commands and control information.

The next paragraphs present an implementation of the IBP communication functions, written in assembly language.

```
SOKE      EQU  50h
SOKE_CO   EQU  55h
ST_VGA    EQU  0b800h
screen_ecb: ;event control block for image packets
          dw 2 dup(0)          ;link address (internal)
          dw 2 dup(0)          ;event service routine (none)
in_use    db 0                 ;in use flag
          db 0                 ;completion code
          dw SOKE              ;image socket 5000h (hi lo)
          db 16 dup(0)         ;workspace (reserved)
          db 6 dup(0ffh)       ;immediate address = broadcast
          dw 3                 ;fragment count
          dw OFFSET ant_ipx_ec    ;IPX header addr.(OFF SEG)
          dw SEG ant_ipx_ec
          dw 30           ;length of IPX header
ofs_ec    dw 0            ;VGA RAM start addr. (OFF SEG)
          dw ST_VGA
          dw 500          ;length of packets (image fragments)
          dw OFFSET nr_comp  ;nr_comp = number of current
          dw SEG nr_comp     ;image fragment
          dw 2               ;length of nr_comp

ant_ipx_ec: ;IPX header for image packets
          dw 0                  ;checksum
          dw 0                  ;length in bytes of total packet
          db 0                  ;transport  control
          db 0                  ;packet type = unknown
          db 4 dup(0)    ;destination network =current network
```

```
          db 6 dup(0ffh)      ;destination node = broadcast
          dw SOKE             ;image socket(hi lo)
          db 4 dup(0)         ;source network
          db 6 dup(0)         ;source node
          dw SOKE             ;image socket(hi lo)

com_ecb: ;command event control block
          dw 2 dup(0)         ;link address (internal)
          dw OFFSET rut_term ;event service routine(OFF SEG)
          dw SEG rut_term     ;the event service routine takes
;care of the registration process (in the master module), or
;executes the master's commands (in the slave module)
in_use_co db 2 dup(0)         ;in use flag + completion code
          dw SOKE_CO          ;command socket 5500h (hi lo)
          db 16 dup(0)        ;workspace (reserved)
com_add   db 6 dup(0ffh)      ;immediate address = broadcast
;some of the commands are broadcast, but some require a
;precise destination
          dw 2                ;fragment count
          dw OFFSET ant_ipx_co ;com. IPX header addr.(OFF SEG)
          dw SEG ant_ipx_co
          dw 30               ;length of IPX header
          dw OFFSET comanda   ;command buffer add.(OFF SEG)
          dw SEG comanda
          dw 50 dup(?)        ;length of command buffer

ant_ipx_co:
          db 10 dup(0)   ;checksum+length+tr.cont.+type+dest.
com_dest  db 6 dup(0ffh);dest. node=broadcast (not for all)
          dw SOKE_CO     ;command socket (hi lo)
          db 4 dup(0)    ;source network
com_src   db 6 dup(0)    ;source node
          dw SOKE_CO     ;command socket (hi lo)

nr_comp   dw   0 ;keeps the image fragment number
comanda   db 50 dup(?) ;keeps commands (received or to be sent)
```

```
trans: ;packet transmission
     mov       bx,3      ;function code (3=transmision)
     push      cs
     pop       es
     mov       si,OFFSET screen_ecb    ;for image packets, or
                         ;OFFSET com_ecb for command packets
     int       7ah
     ret

listen: ;listening for packets
     mov       bx,4      ;function code (4=reception)
     push      cs
     pop       es
     mov       si,OFFSET screen_ecb    ;or OFFSET com_ecb
     int       7ah
     ret
```

The "trans" and "listen" functions return immediately, and IPX carries out the operations in background.

Because the length of an IPX package is limited to 546 bytes, the screen images that are 4000 bytes long in 40x80 text modes, must be divided in fragments small enough to match the IPX requirements. For example we can form 8 packages of 500 bytes for every screen image. The transfer of such a packet through a 10 Mega Bits/Sec network segment, takes approximately 0,4 ms, and the whole image can be transferred in about 3,2 ms. Dynamic images must be reproduced with a refreshing rate of about 30 frames per second. A higher rate unnecessary crowds the communication media, and a slower rate looks bad on screens. The real time clock (RTC) can be used to uniformly broadcast the image packages, by initiating a transmission at every clock tic. The period of the RTC must be shortened in order to obtain the recommended refresh rate. The next example shows how the RTC frequency can be modified.

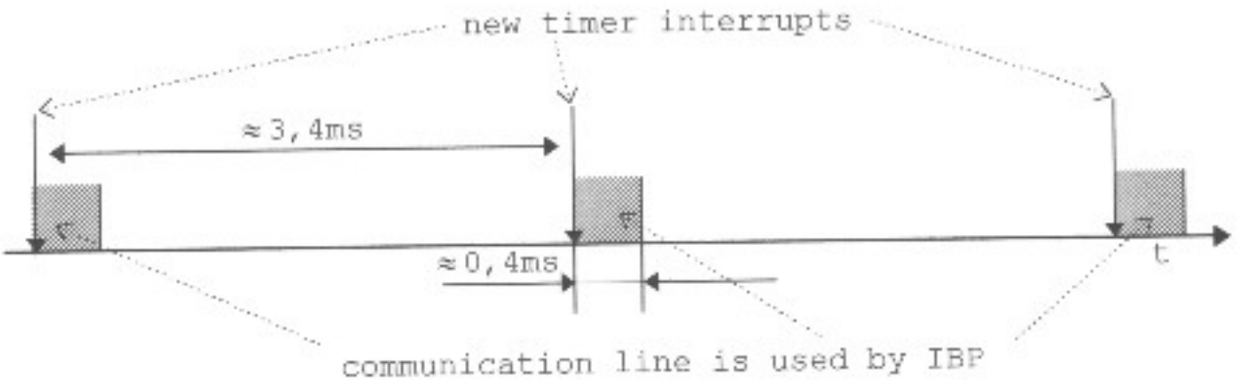
```
    cli
    mov  al,00110110B        ;command to load the counter
    out  43h,al              ;43h=command port
    mov  ax,1000h            ;counter=original value/16
    out  40h,al              ;40h=data port
    mov  al,ah
    out  40h,al
    sti
```

This modification must not affect the rest of the system. The problem can be solved by overriding the timer interrupt (int 8), and by calling the original service routine at every 16 new timer ticks.

```
cont db 0            ;interrupt counter
tim_adr dw 2dup(?) ;holds the original service routine
new_clock: ;the new int 8 (timer) service routine
    call salv_context  ;saves CPU registers.
    inc  cont          ;increments the interrupt counter
    and  cont,0fh
    jnz  fara_orig
    pushf
    call DWORD PTR cs:[tim_adr]  ;the original int 8 service
    jmp  achitata
fara_orig:
    mov  al,20h        ;acknowledge the interrupt
    out  20h,al
achitata:
    call trans         ;sends a packet
    call ref_context   ;restores CPU registers
    iret
```

The next figure presents the loading level of the communication media, when using the IBP (command packets are rarely needed, and can be ignored). The IBP uses about 12% of the communication media capacity.

fig.5. IBP's communication timings



For taking over the operator's commands, the master module overrides the keyboard interrupt (int 9).

The next section presents another implementation of the IBP communication routines, using the C language, and the Novell C API library.

```
#include <stdio.h>
#include "nxt.h"
#define TMP_SOC (BYTE) 0          //temporary socket
#define MEM_VIDEO 0xB800          //start of VGA RAM
ECB ecb;                          //Event Control Block
IPXHeader antetIPX;               //IPX header
WORD nrSoclu = 0x50;              //socket number (hi lo)
BYTE adrNod[6]={0xff,0xff,0xff,0xff,0xff,0xff}; //broadcast
BYTE adrRetea[4]={0,0,0,0};       //current network
BYTE far *adrEcran;               //image fragment address
int init_IPX()
{
     int status;
     if(status=IPXinitialize()){ //initialise IPX driver
          printf("\nerror=%x, initialising IPX\n",status);
          return status;
     }                            //open temporary socket
     if(status=IPXOpenSocket((void*)&nrSoclu, TMP_SOC)){
          printf("\neroare=%x, opening socket\n",status);
          return status;
```

```
    }
    ecb.ESRAddress = 0;          //initialise ECB
    ecb.inUseFlag = 0;
    ecb.socketNumber=nrSoclu;
    ecb.fragmentCount = 2;
    ecb.fragmentDescriptor[0].address=&antetIPX;
    ecb.fragmentDescriptor[0].size=sizeof(antetIPX);
    movmem(adrNod, ecb.immediateAddress, 6);
    movmem(adrRetea, antetIPX.destination.network, 4);
    movmem(adrNod, antetIPX.destination.node, 6);
    *antetIPX.destination.socket=nrSoclu;
    return 0;
}
void listen(int nr_pac) //receives a packet
{                       //nr_pac = number of the image fragment
    adrEcran = MK_FP(MEM_VIDEO,500*nr_pac);
    ecb.fragmentDescriptor[1].address=adrEcran;
    ecb.fragmentDescriptor[1].size=500;
    IPXListenForPacket(&ecb);
    while(ecb.inUseFlag)
        IPXRelinquishControl();
}
void send(int nr_pac)   //initiates a packet transmission
{
    adrEcran = MK_FP(MEM_VIDEO,500*nr_pac);
    ecb.fragmentDescriptor[1].address=adrEcran;
    ecb.fragmentDescriptor[1].size=500;
    IPXSendPacket(&ecb);
}
void terminate()        //closes the socket
{
    IPXCancelEvent(&ecb);
    IPXCloseSocket(nrSoclu);
}
```

REFERENCES

1. Novell Inc. "NetWare C Interface for MS DOS", 1989;

2. "Novell NetWare LOW LEVEL API Notes";

3. Valentin Cristea , Teora "RETELE DE CALCULATOARE", 1992.

Received at: 02.09.1996

Universitatea din Baia-Mare
Facultatea de Litere şI Ştiinţe
str. Victoriei nr.76
RO-4800 Baia-Mare
ROMANIA