460 Lab Assignment #1

```
                   CS460 Lab Assignment #1
#############################################################################
                     DUE : WATCH date
#############################################################################

1. REQURIEMENT:
   To develop a boot program for booting the MTX operating system.

2. Background Reading List
   Notes #2: Booting

3-1. Download the MTX image file
         http://www.eecs.wsu.edu/~cs460/samples/LAB1/mtximage
     Use it as VIRTUAL FD, as in
        qemu-system-i386 -fda mtximage -no-fd-bootchk
     Then, boot up MTX from the virtual FD.

     Test run the MTX operating system as demonstrated in class.

                         CONTENTS of the MTX disk image:

     |  B0  | B1 ...................................................... B1339 |
     --------------------------------------------------------------------------
     |booter|   An EXT2 file system for MTX; kernel=/boot/mtx                  |
     --------------------------------------------------------------------------

     LAB#1 IS FOR YOU TO WRITE A BOOTER PROGRAM TO REPLACE THE booter IN BLOCK#0
     TO BOOT UP THE MTX KERNEL, which is the image file /boot/mtx.

3-2. Background: Computer Architecture and Programming Environment
     Lab#1 assumes the following hardware and software environments.

     Hardware: Intel X86 based PC running Linux. For convenience, use a virtual
               machine that emulates the PC hardware: QEMU, VMware, VirtualBox,
               etc.
     Software: BCC compiler-assembler-linker under Linux.

     When a PC starts, it is in the so-called 16-bit UNPROTECTED mode, also
     known as the 16-bit real mode. While in this mode, the PC's CPU can only
     execute 16-bit code and access 1MB memory. The diagram below shows the 1MB
     memory layout, shown in 64KB segments.

                      1MB MEMORY layout in 64KB SEGMENTS

           0x0000    0x1000   .......     0x9000     0xA000 ... 0xF000
           -----------------------------------------------------------
           |         |     .........     |          |          BIOS |
           -----------------------------------------------------------
           |<-------------- 640KB RAM area --------->|<--- ROM ----->|

     The CPU's internal registers are
              segment registers: CS, DS, SS, ES
              general registers: AX, BX, CX, DX, BP, SI, DI
              status  register : FLAG
              stack pointer    : SP
              instruction point or program counter: IP
     All registers are 16-bit wide.

     The CPU operates as follows:

1. In real-mode, the CPU has 20-bit address lines for 2**20 = 1MB memory, e.g.
   20-bit addresses
                     0x00000, 0x00010, 0x00020
                     0x10000, 0x20000, 0x30000, etc.

   A segment is a block of memory beginning from a 16-byte boundary. Since the
   last 4 bits of a segment address are always 0, it suffices to represent a
   segment address by the leading 16 bits. Each segment size is up to 64KB.

2. The CPU in has 4 segment registers, each 16-bits.

       CS -> Code  segment  = program code or instructions
       DS -> Data  segment  = static and global data (ONE COPY only)
       SS -> Stack segment  = stack area for calling and local variables.
       ES -> Extra segment  = temp area; may be used for malloc()/mfree()

3. In a program, every address is a 16-bit VIRTUAL address (VA). For each 16-bit
   VA, the CPU automatically translates it into a 20-bit PHYSICAL address (PA)
   by
          (20-bit)PA = ((16-bit)segmentRegister << 4) + (16-bit)VA.

  where segmentRegister is either by default or by a segment prefix in the
  instruction.
  Examples:
  Assume CS=0x1234. IP=0x2345 ==> PA = (0x1234<<4) + 0x2345 = 0x14685 (20 bits)
         DS=0x1000. mov ax,0x1234 ==> PA=0x10000 + 0x1234   = 0x11234, etc.

IMPORTANT: In a program, every address is a 16-bit VA, which is an OFFSET in a
           memory segment. When accessing memory, the CPU maps every VA to a
           20-bit PA.

4. The number of DISTINCT segments available to the CPU depends on the
   memory model of the executing program, which is determined by the compiler
   and linker used to generate the binary executable image. The most often used
   memory models are

    One-segment model :(COM files): CS=DS=SS all in ONE segment <= 64KB
    Separate I&D model:(EXE files): CS=CodeSegment, DS=SS=Data+Stack segment

One-segment model programs can be loaded to, and executed from, any available
segment in memory. In order to run a One-segment memory model program, the
following steps are needed:

(1). A C compiler and assembler which generate 16-bit (object) code
(2). A linker that combines the object code to generate a ONE-segment
     binary executable image. We shall use BCC under Linux to do (1) and (2).
(3). LOAD the binary executable image into memory (at a segment boundary)
     and set CPU's CS=DS=SS = loaded segment.
         Set SP at the HIGH end of the segment.
         Set IP at the beginning instruction in the segment.
     Then let the CPU execute the image.


5. PRE-WORK #1: DUE: in one week

5.1. Download files from samples/LAB1/LAB1.1/

Given: The following bs.s file in BCC's assembly

!=================== bs.s file ==================================
        .globl _main             ! IMPORT symbols from C code
        .globl _getc,_putc       ! EXPORT symbols to C code

!---------------------------------------------------------------------
! Only one SECTOR loaded at (0000,7C00). Load entire block to 0x90000
!---------------------------------------------------------------------
        mov  ax,#0x9000    ! set ES to 0x9000
        mov  es,ax

        xor  bx,bx          ! clear BX = 0
!---------------------------------------------------------------------
!  Call BIOS INT-13 to read BOOT BLOCK to (segment,offset)=(0x9000,0)
!---------------------------------------------------------------------
        xor  dx,dx          ! DH=head=0,   DL=drive=0
        xor  cx,cx          ! CL=cylinder, CL=sector
        incb cl             ! BIOS counts sector from 1
        mov  ax, #0x0202    ! AH=READ      AL=2 sectors
        int  0x13           ! call BIOS INT-13

        jmpi start,0x9000   ! CS=0x9000, IP=start

start:
        mov  ax,cs          ! Set segment registers for CPU
        mov  ds,ax          ! we know ES,CS=0x9000. Let DS=CS
        mov  ss,ax          ! SS = CS ===> all point at 0x9000
        mov  es,ax
        mov  sp,#8192       ! SP = 8192 above SS=0x9000

!--------------- OPTIONAL -------------------------------------
        mov  ax,#0x0012     ! Call BIOS for 640x480 color mode
        int  0x10           !
!---------------------------------------------------------------
        call _main          ! call main() in C

        jmpi 0,0x1000


!======================== I/O functions =================================

!---------------------------------------------
!  char getc()   function: returns a char
!---------------------------------------------
_getc:
        xorb   ah,ah           ! clear ah
        int    0x16            ! call BIOS to get a char in AX
        ret

!----------------------------------------------
! void putc(char c)  function: print a char
!----------------------------------------------
_putc:
        push   bp
        mov    bp,sp

        movb   al,4[bp]        ! get the char into aL
        movb   ah,#14          ! aH = 14
        movb   bl,#0x0D        ! bL = color
        int    0x10            ! call BIOS to display the char

        pop    bp
        ret

Write YOUR own t.c file in C:

/************** t.c file **************************/
int prints(char *s)
{
    call putc(c) to print string s;
}

int gets(char s[ ])
{
    call getc() to input a string into s[ ]
}

main()
{
   char name[64];
   while(1){
     prints("What's your name? ");
     gets(name);
     if (name[0]==0)
        break;
     prints("Welcome "); prints(name); prints("\n\r");
   }
   prints("return to assembly and hang\n\r");
}

5-3. Use BCC to generate a one-segment binary executable a.out WITHOUT header

     as86 -o bs.o  bs.s
     bcc -c -ansi  t.c
     ld86 -d bs.o t.o /usr/lib/bcc/libc.a

5-4. dump a.out to a VIRTUAL FD disk:

     dd if=a.out of=mtximage bs=1024 count=1 conv=notrunc

5-5. Boot up QEMU from the virtual FD disk:

     qemu-system-i386 -fda mtximage -no-fd-bootchk

5-6. For YOUR benefit: do ALL steps of 5-3 to 5-5 by a sh script.


6. PRE-WORK #2: DUE: in ONE week
   Download files from samples/LAB1/LAB1.2

   mtximage contains an EXT2 file system, block size = 1KB (1024 bytes)

     |  B0  |  B1   B2....................       ..................... B1339 |
     -------------------------------------------------------------------------
     |booter| SUPER|GD |Bmap|Imap|INODES                                |      |
     -------------------------------------------------------------------------

   Write your C code to print all the file names in the root directory /

             The complete bs.s file:
==========================================================================
        BOOTSEG =  0x9000        ! Boot block is loaded again to here.
        SSP      =   8192        ! Stack pointer at SS+8KB

        .globl _main,_prints              ! IMPORT symbols
        .globl _getc,_putc                ! EXPORT symbols

        .globl _readfd,_setes,_inces,_error

        !-------------------------------------------------------
        ! Only one SECTOR loaded at (0000,7C00). Get entire BLOCK in
        !-------------------------------------------------------
        mov  ax,#BOOTSEG    ! set ES to 0x9000
        mov  es,ax
        xor  bx,bx          ! clear BX = 0

        !---------------------------------------------------
        !  call BIOS to read boot BLOCK to [0x9000,0]
        !---------------------------------------------------
        xor  dx,dx          ! drive 0, head 0
        xor  cx,cx
        incb cl             ! cyl 0, sector 1
        mov  ax, #0x0202    ! READ 1 block
        int  0x13

        jmpi    start,BOOTSEG           ! CS=BOOTSEG, IP=start

start:
        mov     ax,cs                   ! Set segment registers for CPU
        mov     ds,ax                   ! we know ES,CS=0x9000. Let DS=CS
        mov     ss,ax                   ! SS = CS ===> all point at 0x9000
        mov     es,ax
        mov     sp,#SSP                 ! SP = 8KB above SS=0x9000

        mov     ax,#0x0012              ! 640x480 color
        int     0x10

        call    _main                   ! call main() in C

        test    ax,ax                   ! main() return 1 for OK, 0 for BAD
        je      _error                  ! if main() return 0, jump to _error

        jmpi 0,0x1000

!======================== I/O functions =================================

        !---------------------------------------------
        !  char getc()   function: returns a char
        !---------------------------------------------
_getc:
        xorb   ah,ah           ! clear ah
        int    0x16            ! call BIOS to get a char in AX
        ret

        !----------------------------------------------
        ! void putc(char c)  function: print a char
        !----------------------------------------------
_putc:
        push   bp
        mov    bp,sp

        movb   al,4[bp]        ! get the char into aL
        movb   ah,#14          ! aH = 14
        movb   bl,#0x0D        ! bL = cyan color
        int    0x10            ! call BIOS to display the char

        pop    bp
        ret

       !----------------------------------------
       ! readfd(cyl, head, sector, buf)
       !         4     6     8     10    byte offset from stack frame pointer bp
       !----------------------------------------
_readfd:
        push  bp
        mov   bp,sp            ! bp = stack frame pointer

        movb  dl, #0x00        ! drive in DL: 0=FD0
        movb  dh, 6[bp]        ! head in  DH
        movb  cl, 8[bp]        ! sector in CL
        incb  cl               ! BIOS count sector from 1
        movb  ch, 4[bp]        ! cyl in CH
        mov   bx, 10[bp]       ! BX=buf ==> loading memory addr=(ES,BX)
        mov   ax, #0x0202      ! AH=READ, AL=2 sectors to (EX, BX)

        int  0x13              ! call BIOS 0x13 to read the block
        jb   _error            ! to error if CarryBit is on [read failed]

        pop  bp
        ret

_setes: push  bp
        mov   bp,sp

        mov   ax,4[bp]
        mov   es,ax

        pop   bp
        ret

_inces:                          ! inces() inc ES segment by 0x40, or 1KB
        mov   ax,es
        add   ax,#0x40
        mov   es,ax
        ret

        !------------------------------
        !       error & reboot
        !------------------------------
_error:
        mov  bx, #bad
        push bx
        call _prints

        int  0x19                       ! reboot

bad:    .asciz  "Error!"
====================================================================
The getblk(u16 blk, char *buf) function:

int getblk(u16 blk, char *buf)
{
    readfd( (2*blk)/CYL, ( (2*blk)%CYL)/TRK, ((2*blk)%CYL)%TRK, buf);
}

The booter runs in the segment 0x9000.
CS, DS, SS, ES are all pointing at the same segment 0x9000.
BIOS will load the disk block blk to (ES, buf), which is the char buf[ ] in
the program.
==========================================================================

7. Modify YOUR t.c in LAB1.2 to get the INODE of /boot/mtx, i.e.

   let    INODE *ip -> INODE of /boot/mtx

   ip->i_blokc[0] to ip->i_block[11] are DIRECT blocks

   ip->i_block[12] = IDIRECT block = an array of 32-bit integers, 0 if no more.

To load the blocks into memory starting from (segment) 0x1000

   setes(0x1000);          // ES = 0x1000 => BIOS loads disk block to (ES, buf)

loop:
   getblk((u16)blkno, 0);     // buf = 0 => memory address = (ES, 0)
   inces();                   // inc ES by 1KB/16 = 0x40
repeat loop for next blkno until no more


8. LAB1 main work: DUE: in 2 weeks

   Modify t.c file to boot up /boot/mtx from the mtximage VIRTUAL disk

sample solution: mtximage  (run qemu on it)
```