Study Material for File Operation using int86,int86x,intdos,intdosx functions

```
/*====================================================================
 Assignment 5   (using intdosx,intdos)
 Title :        copy a file,Delete a file, Create a directory

====================================================================*/
```

intdosx()
Set Seg Regs and Invoke DOS Function

```
#include   <dos.h>
```

```
int       intdosx(inregs,outregs,segregs);
union REGS    *inregs;                Register values going into call
union REGS    *outregs;               Register values on return
struct SREGS  *segregs;               ES and DS values going into call
```

intdosx() invokes a DOS system call (Interrupt 0x21) after setting the registers to the values in 'inregs' and setting the DS and ES registers to the values in 'segregs'. 'outregs' is set to the value of the registers after the system call. The 'cflag' field of 'outregs' is set to the status of the system carry flag; a non-zero value indicates a DOS error. 'inregs.h.ah' is the DOS function number.

Returns: The value of the AX register after the system call. If the 'cflag' field in 'outregs' is non-zero, an error has occurred and '_doserrno' (defined in <errno.h>) is set to the error code.

Notes: The values for segment registers can be obtained with the segread() function or the FP_SEG() macro.

If the DS and ES registers don't need to be set, intdos() can be used.

DOS functions that use only the DX and AL registers and that don't return error information in the carry flag can be invoked via the bdos() or the bdosptr() function. Note that neither allow the setting of the segment register ES.

Portability: MS-DOS only.

---------- **Example** ----------

The following statements output 'string' to the standard output.

```
#include <dos.h>      /* intdosx(), macros FP_SEG() and FP_OFF(),
                         union REGS and struct SREGS */

union REGS inregs, outregs;
struct SREGS segregs;
```

```
char far *string =
             "this string is not in the default data segment$";

main()
{
    inregs.h.ah = 0x9;         /* output string function number */
    inregs.x.dx = FP_OFF(string);/*DS:DX is address of 'string*/
    segregs.ds = FP_SEG(string);
    intdosx(&inregs, &outregs, &segregs);
}
```

int86x()

Set Segment Registers and Execute Software Interrupt

```
#include   <dos.h>
```

int int86x(intno,inregs,outregs,segregs);

```
int           intno;                Interrupt number
union  REGS   *inregs;              Register values going into call
union  REGS   *outregs;             Register values on return
struct SREGS  *segregs;             Segment-register values on call
```

int86x() executes the 8086 software interrupt 'intno'; it is similar to int86(), but it allows you to set the DS and ES segment registers also. The registers are set to the values in 'inregs'; the DS and ES segment registers are set to the corresponding values in 'segregs'. On return, 'outregs' is set to the value of the registers after the interrupt has executed; DS is restored; and the 'cflag' field of 'outregs' is set to the status of the system carry flag.

Returns: The value of the AX register after the system call. If the 'cflag' field in 'outregs' is non-zero, an error has occurred and '_doserrno' (defined in <stdlib.h>) is set to the error code.

Notes: If the DS and ES registers don't need to be set, int86() should be used. The values for the segment registers can be obtained with the segread() function or the FP_SEG macro.

Use bdos(), intdos() and intdosx() to execute DOS system calls (interrupt 21h).

The DS register is restored upon completion of the int86x() call

---------- **Example** ----------

The following statements read the boot sector (track 0, sector 1) of the disk in drive A: by calling BIOS interrupt 13h.

```
#include <dos.h>     /* for int86x(), macros FP_SEG() and FP_OFF(),
                        segread(), union REGS and struct SREGS */
#include <stdio.h>   /* for printf() */
#include <stdlib.h>  /* for _doserrno */

char buf [1024];

main()
```

```
{
    char far * bufptr;
    union REGS inregs, outregs;
    struct SREGS segregs;

    segread(&segregs);           /* set the segment registers */
    bufptr = (char far *) buf;   /* need a far pointer to 'buf' */
    segregs.es = FP_SEG(bufptr); /* ...to set the address of */
    inregs.x.bx = FP_OFF(bufptr);/* ...'buf' for the BIOS */
    inregs.h.ah = 2;             /* set the BIOS function number */
    inregs.h.al = 1;             /* set # of sectors to read */
    inregs.h.ch = 0;             /* set track # of boot sector */
    inregs.h.cl = 1;             /* set sector # of boot sector */
    inregs.h.dh = 0;             /* set disk side number */
    inregs.h.dl = 0;             /* set drive number to A: */
    int86x(0x13, &inregs, &outregs, &segregs);  /* read sector */

    if (outregs.x.cflag)
        printf("ERROR #%d: status = %d, # sectors read = %d\n",
               _doserrno, outregs.h.ah, outregs.h.al);
}
```

Interrupt Handling with C

C is sometimes called a high level assembly language because it can call the different interrupts using some of its some defined functions. Some important functions are as follows:

- int86: Invokes MS-DOS interrupts.
- int86x: Invokes MS-DOS interrupt with segment register values.
- intdos: invokes MS-DOS service using registers other than DX and AL
- intdosx: invokes MS-DOS service with segment register values.
- segread: Reads Segment registers

We shall discuss these functions in detail. First of all we discuss some predefined structure and unions that are frequently or necessarily used with these functions.

SREGS Structure

This structure has been defined in dos.h and it is a structure of the segment registers passed to and filled in by the functions, int86x, intdosx and segread. The declaration of the structure is as follows:

struct SREGS {
unsigned int es;
unsigned int cs;
unsigned int ss;
unsigned int ds;
};

REGS union

REGS is the union of two structures. The union REGS has been defined dos.h and it is used to pass information to and from the functions, int86, int86x, intdos and intdosx. The declaration of the union is as follows:

union REGS {
struct WORDREGS x;
struct BYTEREGS h;

};

BYTEREGS and WORDREGS Structures

The BYTEREGES and WORDREGS structures have been defined in dos.h and these are used for storing byte and word registers. The WORGREGS structure allows the user to access the registers of CPU as 16-bit quantities where BYTEREGES structure gives the access to the individual 8-bit registers.

The BITEREGS structure is declared as follows:

struct BYTEREGS {
unsigned char al, ah, bl, bh;
unsigned char cl, ch, dl, dh;
};
And the WORDREGS structure is declared as follows:

struct WORDREGS {
unsigned int ax, bx, cx, dx;
unsigned int si, di, cflag, flags;
};

int86() Execute 8086 Software Interrupt

```
#include   <dos.h>
```

int int86(intno,inregs,outregs);

```
int         intno;                  Interrupt number
union REGS  *inregs;                Register values going into call
union REGS  *outregs;               Register values on return
```

```
   int86() executes the 8086 software interrupt 'intno'. The registers
   are set to the values in 'inregs' before the interrupt is executed.
   'outregs' is set to the value of the registers after the interrupt is
   executed.  The 'cflag' field of 'outregs' is set to the status of the
   system carry flag.

   Returns:    The value of the AX register after the system call. If
               the 'cflag' field in 'outregs' is non-zero, an error has
               occurred and '_doserrno' (defined in <stdlib.h>) is set
               to the error code.

     Notes:    If 'far' or 'huge' addresses are passed to the interrupt
               routine, int86x() may needed instead of int86().

               DOS interrupts (interrupt #21H) can also be executed with
               intdos().

               The union type REGS is defined in <dos.h>.
```

```
 ------------------------------- Example ---------------------------------
```

The following statements clear the screen by using BIOS interrupt 10H to first read the current video mode, then reset the video mode to its current mode:

```
#include <dos.h>       /* for int86() and union REGS */

main()
{
    union REGS inregs, outregs;

    inregs.h.ah = 0x0F;              /* set BIOS function number */
    int86(0x10, &inregs, &outregs);  /* read video mode */

    inregs.h.ah = 0;                 /* set BIOS function number */
    inregs.h.al = outregs.h.al;      /* set video mode number */
    int86(0x10, &inregs, &outregs);  /* set video mode */
                                     /* also clears the screen) */
}
```