2020/05/01_第22课_文件相关函数的使用、文件缓冲区

笔记本: C
创建时间: 2020/5/1 星期五 16:28
作者: ileemi

- 使用文件操作模拟命令行copy

文本操作基于二进制上面的再次封装

stream(流) ->可理解为设备

C语言中的设备抽象层就是文件，文件的打开，关闭，读写，可以针对任何设备

如果数据在栈空间内：栈内的数据是可以被栈自动维护的，基本不要要频繁的检查，但是如果数据出了栈结构，对文件进行操作，一步一检查

文件函数的使用：

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    FILE *fp = NULL;
    fp = fopen("Test.bin", "rb+");
    //判断文件是否存在，且是否打开成功
    if (fp == NULL)
    {
        //如果文件不存在，创建一个对应的文件，并以二进制形式读和写打开文件
        fp = fopen("Test.bin", "wb+");
        if (fp == NULL)
        {
            //再次判断文件是否打开成功
            exit(-1);
        }
    }
    //具体操作
    int nTemp = fputc('a', fp);
    if (nTemp == EOF)
    {
        //error,错误处理
    }
    //在刚才的文件末尾添加新的字符，应将文件指针移动到文件的结尾
    fseek(fp, 0, SEEK_END);
```

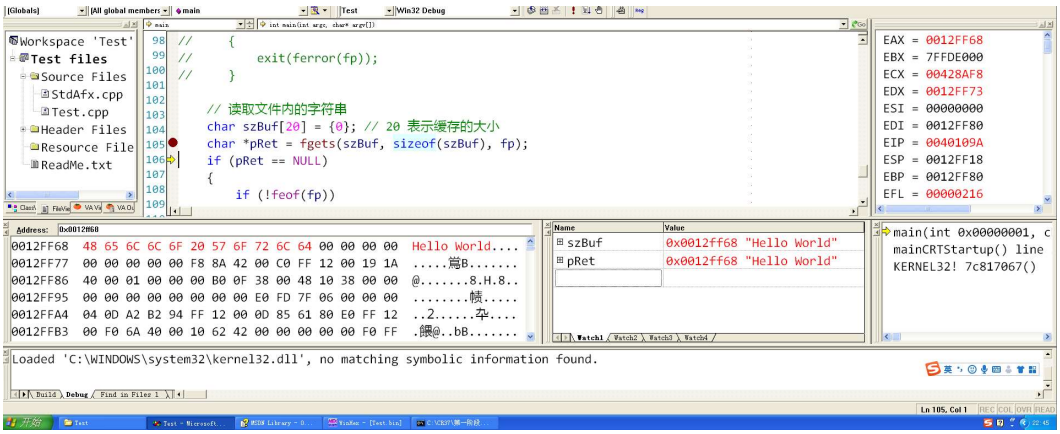
```
    int  nflose  =  fseek(fp,  0,  SEEK_END);
    if  (nflose  !=  0)
    {
        //error,错误处理
    }
    int  nTemp2  =  fputc('b',  fp);
    if  (nTemp2  ==  EOF)
    {
        //error,错误处理
    }
    //在文件关闭前就将数据写入到文件内
    fflush(fp);  //刷新缓冲区
/*
    //使用fgetc  读取文件内的所有字符，并显示到控制台
    char  ch  =  '\0';
    while(ch  =  fgetc(fp))
    {
        if  (ferror(fp)  !=  0)
        {
            //  如果在流上没有发生错误，ferror返回0。否则，它将返回一个
非零值。
            break;
        }
        //判断是否到达文件末尾
        if  (ch  ==  EOF)
        {
            //显示错误信息
            break;
        }
        putchar(ch);
    }
*/
    if  (fp)
    {
        fclose(fp);
        fp  =  NULL;
    }
    system("pause");
    return  0;
}
```

feof 测试是否到达文件末尾

在文件内添加一个字符串，并读出来

```
#include  <stdio.h>
#include  <stdlib.h>
```

```c
int  main(int  argc,  char*  argv[])
{

    FILE  *fp  =  NULL;
    fp  =  fopen("Test.bin",  "rb+");
    //判断文件是否存在，且是否打开成功
    if  (fp  ==  NULL)
    {
        //如果文件不存在，创建一个对应的文件，并以二进制形式读和写打开
文件
        fp  =  fopen("Test.bin",  "wb+");
        if  (fp  ==  NULL)
        {
            //再次判断文件是否打开成功
            exit(-1);
        }
    }

    //int  nRet  =  fseek(fp,  0,  SEEK_END);
    //将文件指针fp  指向该文件的结尾处
    if  (fseek(fp,  0,  SEEK_END))
    {
        exit(ferror(fp));
    }

    //  读取文件内的字符串
    char  szBuf[20]  =  {0};  //  20  表示缓存的大小
    char  *pRet  =  fgets(szBuf,  sizeof(szBuf),  fp);
    if  (pRet  ==  NULL)
    {
        if  (!feof(fp))
        {
            exit(ferror(fp));
        }
    }
    if  (fp)
    {
        fclose(fp);
        fp  =  NULL;
    }
    system("pause");
    return  0;
}
```



fprintf、fscanf函数的使用，代码示例：

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    FILE *fp = NULL;
    fp = fopen("haha.bin", "rb+");
    //判断文件是否存在，且是否打开成功
    if (fp == NULL)
    {
        //如果文件不存在，创建一个对应的文件，并以二进制形式读和写打开文件
        fp = fopen("haha.bin", "wb+");
        if (fp == NULL)
        {
            //再次判断文件是否打开成功
            exit(-1);
        }
    }

    //fprintf(fp, "%s\t%d\t%f\r\n", "你好 世界", 999, 3.14f);
    float flt = 0.0f;
    int n;
    char szBuf[20];
    //从文件指针fp所指向的文件按照指定格式读取对应数据
    fscanf(fp,"%f %d %s", &flt, &n, szBuf);
    //关闭文件
    if (fp)
    {
        fclose(fp);
        fp = NULL;
    }
    system("pause");
    return 0;
}
```

fread、fwrite简单使用，代码示例：

```
#include <stdio.h>
#include <stdlib.h>

struct tagTest
{
    struct tagTest *pNext;
    // 字符串指针，当该结构体中的数据被保存到文件中，
    // 这里保存的是一个指针，源数据无法读取
    char *pszName;
    int nAge;
    float fHeight;
};

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

    FILE *fp = NULL;
    fp = fopen("Test.bin", "rb+");
    // 判断文件是否存在，且是否打开成功
    if (fp == NULL)
    {
        // 如果文件不存在，创建一个对应的文件，
        // 并以二进制形式读和写打开文件
        fp = fopen("Test.bin", "wb+");
        if (fp == NULL)
        {
            // 再次判断文件是否打开成功
            exit(-1);
        }
    }

    // 将文件指针移动到末尾处
    if (fseek(fp, 0, SEEK_END))
    {
            exit(ferror(fp));
    }

    // 初始化结构体数据
    /*
    读取数据时，对应的位置是一个指针，保存字符串，
    应该将字符换的内容写入到文件中去 fputs、fwirte
    */
    struct tagTest test = {
        &test,
```

```
        "Tom",
        20,
        186.25f
    };
    // 将结构体中的数据写入到内存中，可使用fwrite
    if (fwrite(&test, sizeof(test), 1, fp) != 1)
    {
        exit(ferror(fp));
    }
    // 关闭文件
    if (fp)
    {
        fclose(fp);
        fp = NULL;
    }
    system("pause");
    return 0;
}
```

使用文件操作模拟命令行copy

代码示例:

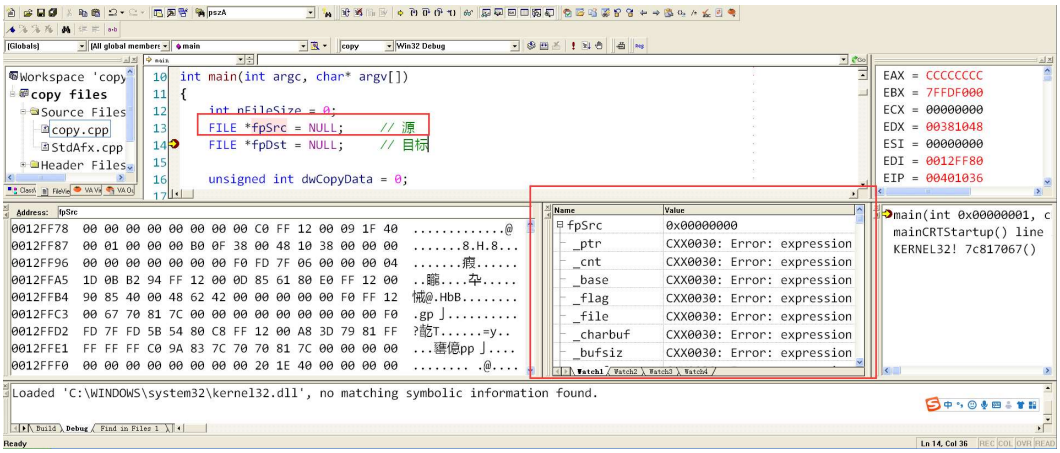
```
#include <stdio.h>
#include <stdlib.h>
#define for if(1)for

int main(int argc, char* argv[])
{
    int nFileSize = 0;
    FILE *fpSrc = NULL;          // 源
    FILE *fpDst = NULL;          // 目标
    unsigned int dwCopyData = 0;
    unsigned char byteCpytData = '\0';
    if (argc < 3)
    {
        printf("命令语法不正确\r\n");
        exit(-1);
    }

    //读取二进制文件必须使用加 b，二进制数据可以处理一切数据，而文本方式只能处理ASCII码串
    fpSrc = fopen(argv[1], "rb+");
    if (fpSrc == NULL)
    {
```

```
            printf("系统找不到指定文件\r\n");
            //  走错误流程
            goto  EXIT_PROC;
    }
    //读取二进制文件必须使用加  b
    fpDst  =  fopen(argv[2],  "wb+");
    if  (fpDst  ==  NULL)
    {
        printf("系统找不到指定文件\r\n");
        //  走错误流程
        goto  EXIT_PROC;
    }
    //将文件指针指向文件的结尾处
    if  (fseek(fpSrc,  0,  SEEK_END)  !=  0)
    {
        //  走错误流程
        goto  EXIT_PROC;
    }
    //  ftell函数告诉我们文件指针的位置，文件指针fpSrc距离文件头部的总
字节数
    nFileSize  =  ftell(fpSrc);
    if  (fseek(fpSrc,  0,  SEEK_SET)  !=  0)
    {
        //  走错误流程
        goto  EXIT_PROC;
    }
    if  (fseek(fpDst,  0,  SEEK_SET)  !=  0)
    {
        //  走错误流程
        goto  EXIT_PROC;
    }
    //  连续4字节进行读取，同时也可以1字节1字节进行读取
    for  (int  i  =  0;  i  <  nFileSize  /  4;  i++)
    {
        //  读取二进制文件不加  b，程序到这里会报：再读取  4096  字节的
时候，程序读取错误
        if  (fread(&dwCopyData,  sizeof(dwCopyData),  1,  fpSrc)  !=  1)
        {
            printf("再读取  %d  字节的时候，程序读取错误
\r\n",ftell(fpSrc));
            goto  EXIT_PROC;
        }
        if  (fwrite(&dwCopyData,  sizeof(dwCopyData),  1,  fpDst)  !=  1)

        {
            printf("再写入  %d  字节的时候，程序读取错误
\r\n",ftell(fpDst));
```

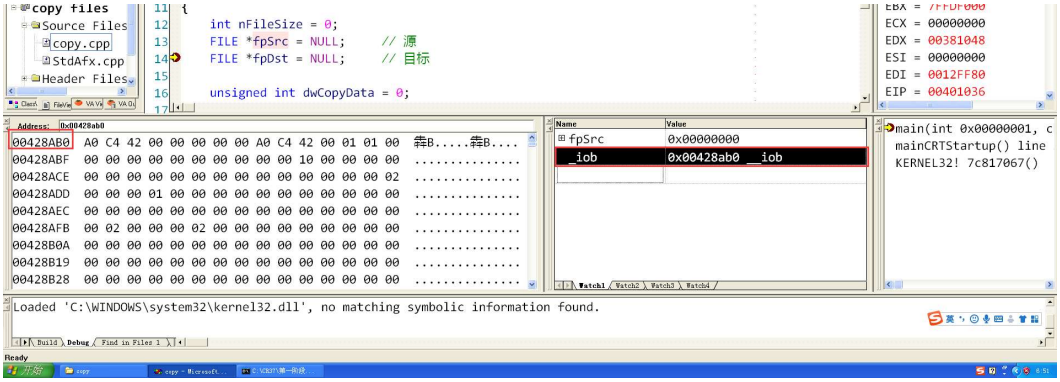
```
            goto  EXIT_PROC;
        }
    }

    for  (int  i  =    0;  i  <  nFileSize  %  4;  i++)
    {
        if  (fread(&byteCpytData,  sizeof(byteCpytData),  1,  fpSrc)  !=  1)

        {
            goto  EXIT_PROC;
        }

        if  (fwrite(&byteCpytData,  sizeof(byteCpytData),  1,  fpDst)  !=

        {
            goto  EXIT_PROC;
        }
    }
    printf("已成功复制  1  个文件\r\n");
    //集中处理错误流程
EXIT_PROC:
    if  (fpSrc)
    {
            fclose(fpSrc);
            fpSrc  =  NULL;
    }
    if  (fpDst)
    {
            fclose(fpDst);
            fpDst  =  NULL;
    }
    system("pause");
    return  0;
}
```

文件缓冲区

文件指针是一个结构体，该结构体对应一些信息，稍后分析



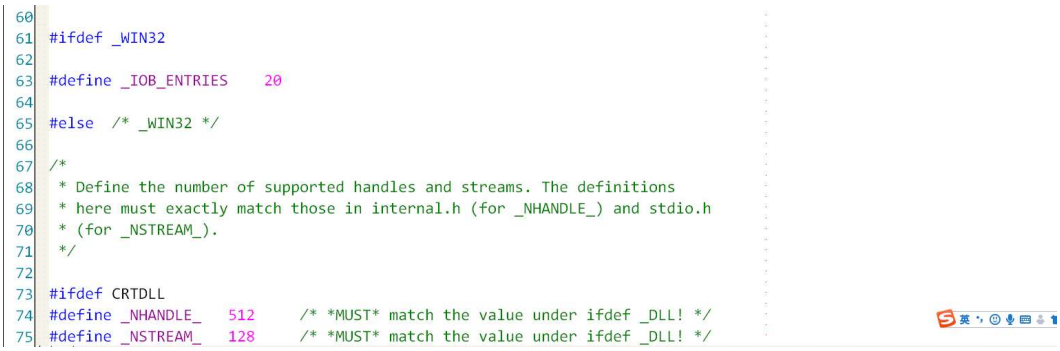
在调试程序时，在当前程序的Watch窗口下，输入_iob，_iob是一个结构体数组。



接着退出调试，我们在程序中，输入_iob，查看其源码

> _iob是一个结构体数组，定义在_FILE.C里面 ， 通过条件编译管理， _iob数组的元素大小可根据环境进行相应的选择

数组成员是一个宏定义： 用来描述_iob数组的size字节大小。



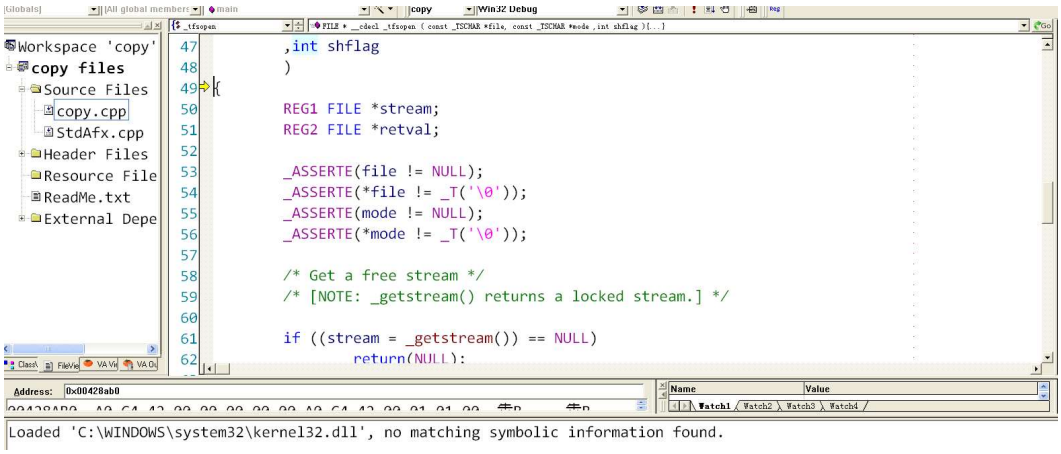
```
/*
 * FILE descriptors; preset for stdin/out/err (note that the __tmpnum field
 * is not initialized)
 */
FILE _iob[_IOB_ENTRIES] = {
        /* _ptr, _cnt, _base,  _flag, _file, _charbuf, _bufsiz */

        /* stdin (_iob[0]) */

        { _bufin, 0, _bufin, _IOREAD | _IOYOURBUF, 0, 0, _INTERNAL_BUFSIZ },

        /* stdout (_iob[1]) */

        { NULL, 0, NULL, _IOWRT, 1, 0, 0 },

        /* stderr (_iob[3]) */

        { NULL, 0, NULL, _IOWRT, 2, 0, 0 },

};
```

_iob 数组前三个元素 已经被定义

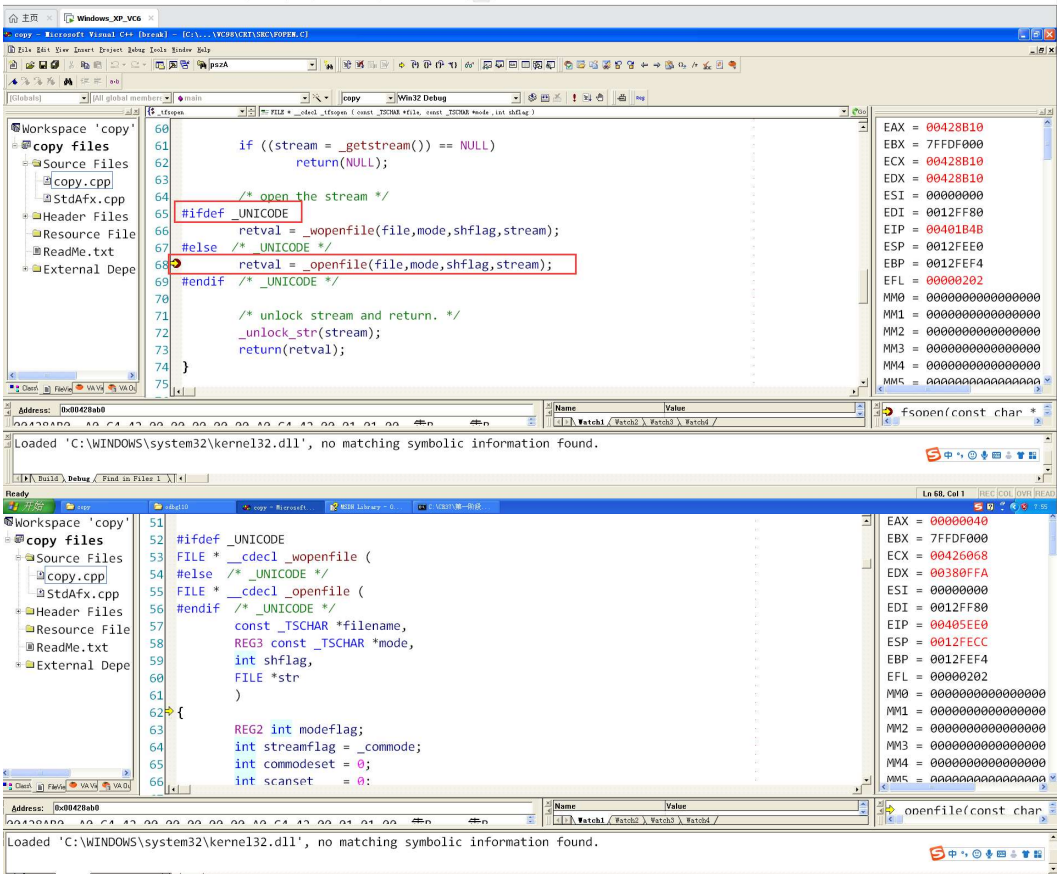
标准错误，不支持重定向到文件，只能输出到显示器

- 调用fopen函数，fopen函数内部调用CreatFile
- 调用fread函数，fread函数内部调用ReadFile 操作系统的winpi接口
- 调用fflush函数，fflush函数内部调用WriteFile

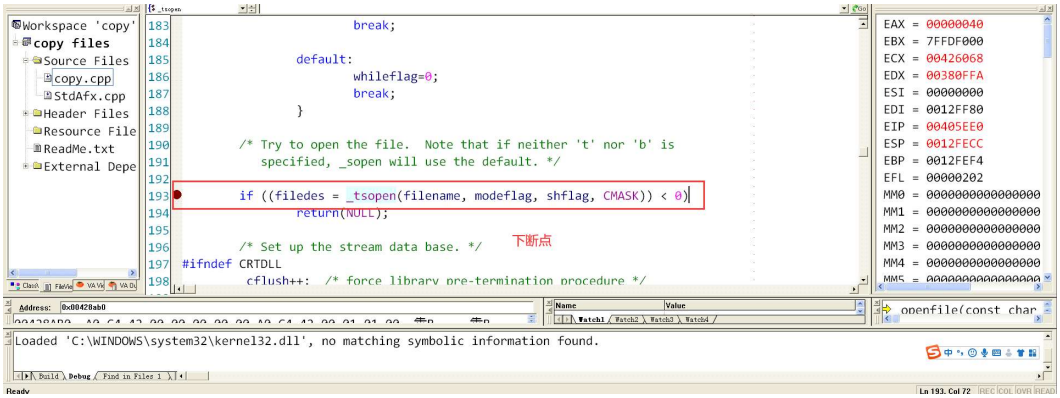
调试程序，当程序运行到 fopen 的时候，F11进入函数内部实现，进入后，继续跟进



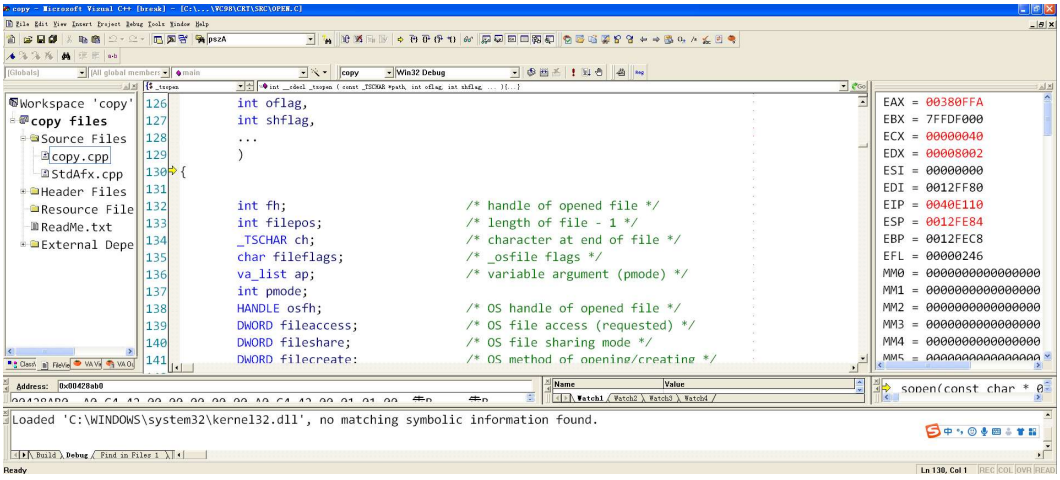
接着根据UNICODE和 非UNICODE选择一个跟进



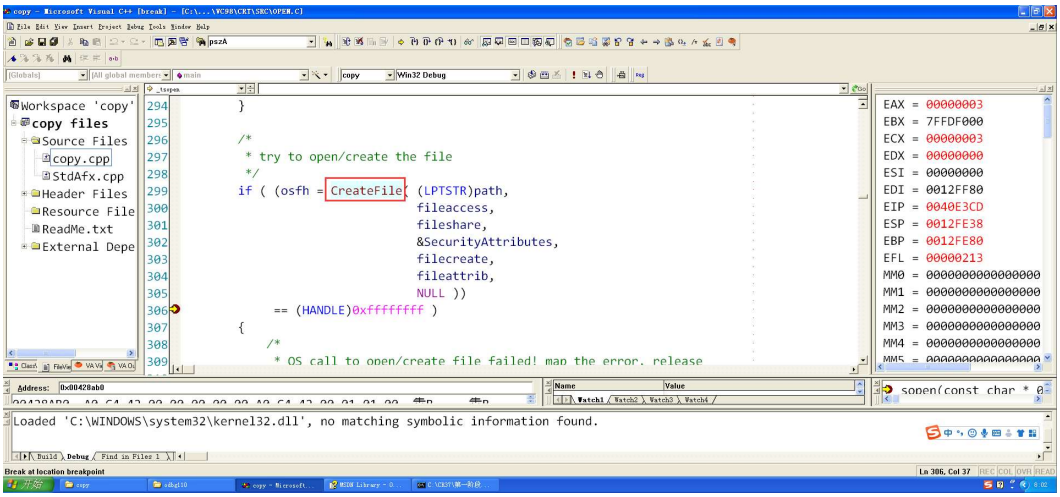
在_tsopen处下断点



然后F11进入函数内部

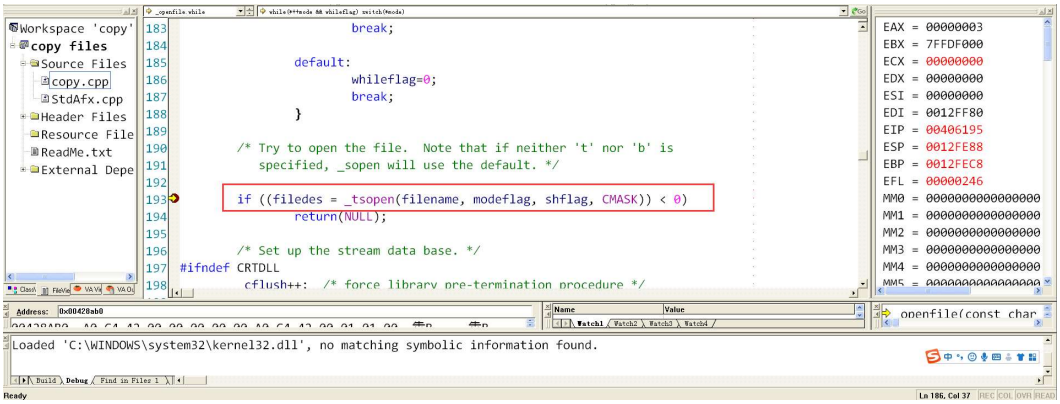


找到CreateFile处，CreateFile打开失败，为-1

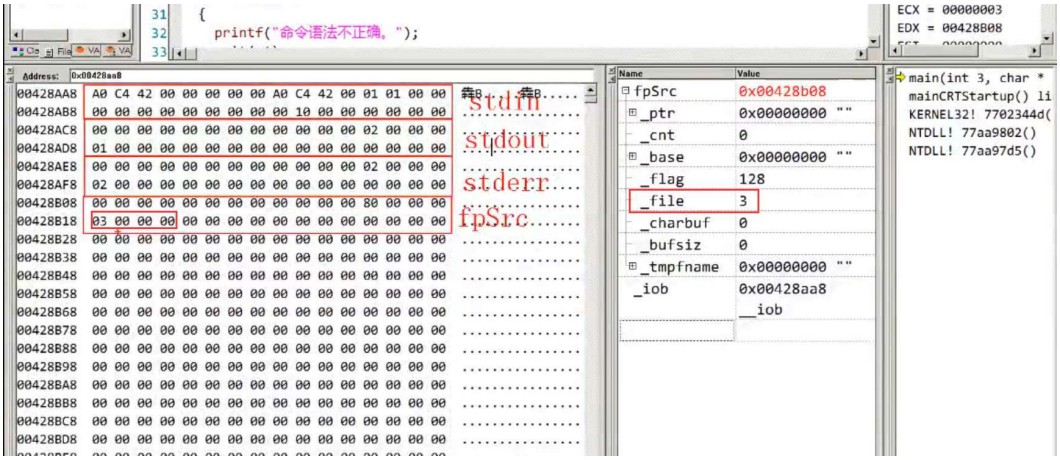


找到关键函数，单步退出 Shift + F11

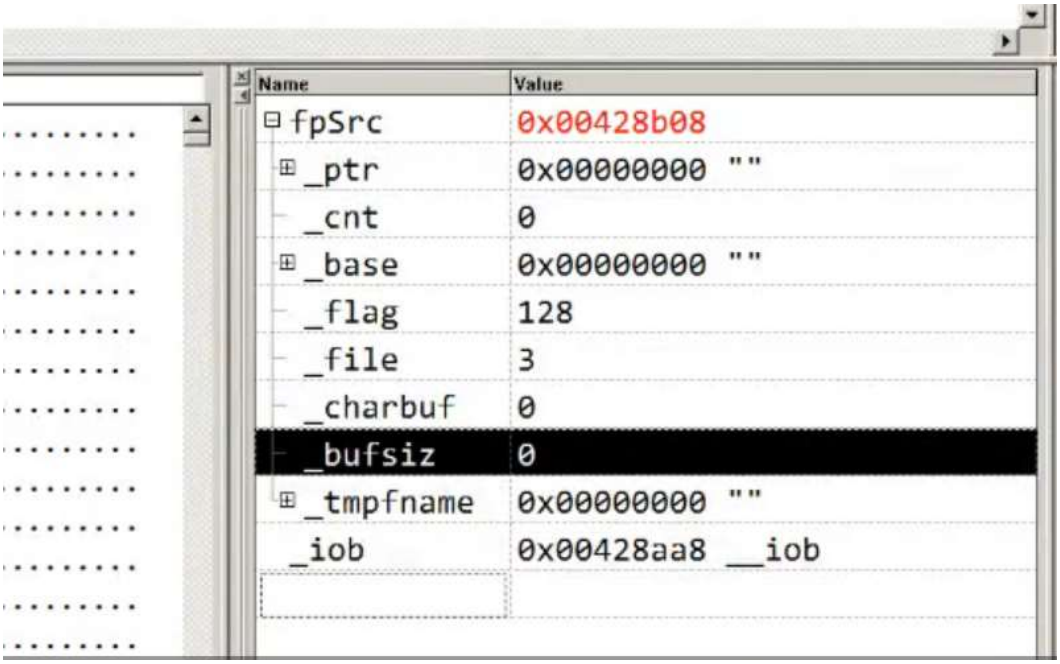
回到_tsopen处



单步退出到main函数内：



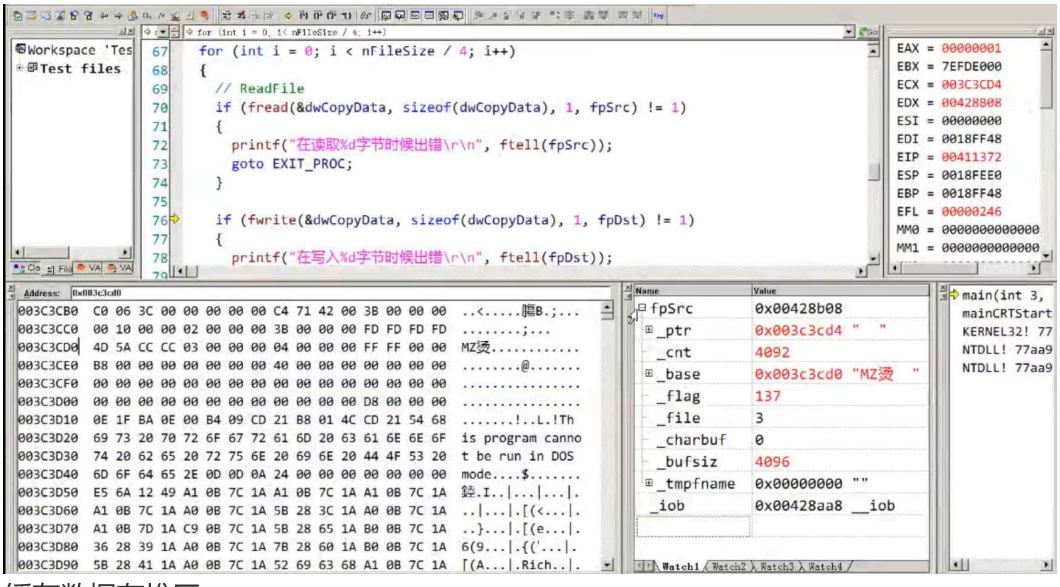
为了减少磁盘的交互，这里做了缓存，把数据在内存中先做好，针对内存做，等用户要提交的时候在讲数据写入到磁盘




```c
struct _iobuf
{
    char *_ptr;  //当前缓冲区内容指针
    int      _cnt;  //缓冲区还有多少个字符
    char *_base;        //缓冲区的起始地址
    int      _flag;        //文件流的状态，是否错误或者结束
    int      _file;        //文件描述符
    int      _charbuf;  //双字节缓冲，缓冲2个字节
    int      _bufsiz;    //缓冲区大小
    char *_tmpfname;//临时文件名
};
typede f struct _iobuf FILE;
```

默认缓存大小：4096 0x1000

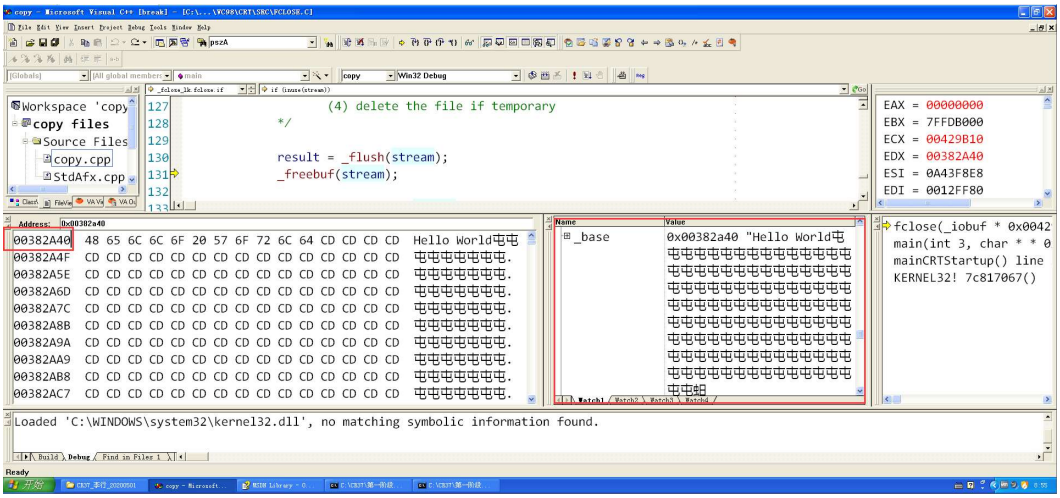
缓存合适开启：当出现第一次读写的时候, 打开文件的时候程序不建立缓存

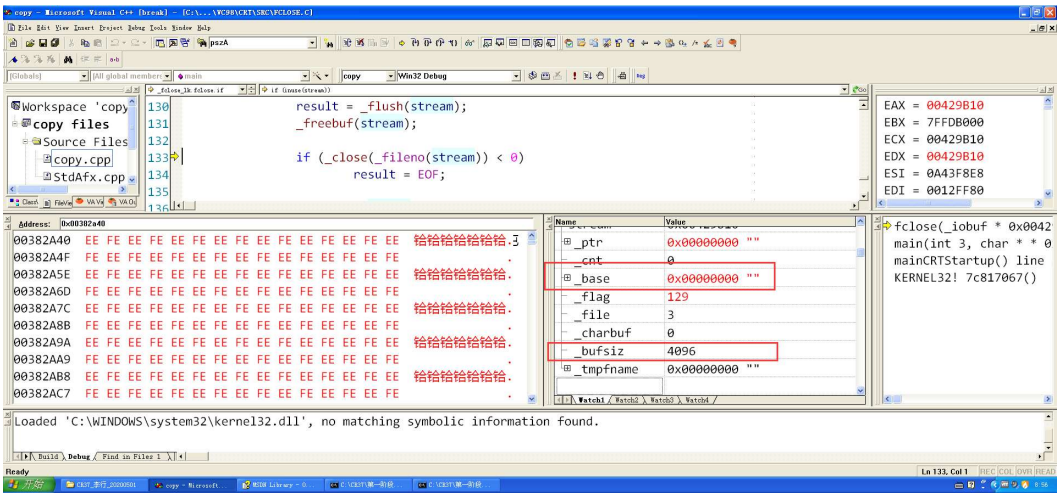


缓存数据在堆区

- 调用fflush函数，fflush函数内部调用WriteFile ， fflush负责写入

关闭文件，fclose 内部调用 Closehandle 函数：





缓存提交不清0，指针回位，_cnt清0

句柄：编号，ID号，唯一标识符