**[linux 标准IO缓冲机制探究](http://blog.csdn.net/wang11chao01/article/details/7787186)**

本文摘自<http://blog.chinaunix.net/uid-26833883-id-3198114.html>，本人从四中所画的图中理解了标准IO和UnBuffered IO的区别。

一、什么是缓存I/O(Buffered I/O)

缓存I/O又被称作标准I/O,大多数文件系统默认I/O操作都是缓存I/O。在Linux的缓存I/O机制中，操作系统会将I/O的数据缓存在文件系统的页缓存(page cache)中，也就是说，数据会先被拷贝到操作系统内核的缓冲区中，然后才会从操作系统内核的缓冲区拷贝到应用程序的地址空间。

1.缓存I/O有以下优点:

A.缓存I/O使用了操作系统内核缓冲区，在一定程度上分离了应用程序空间和实际的物理设备。

B.缓存I/O可以减少读盘的次数，从而提高性能

        当应用程序尝试读取某块数据的时候，如果这块数据已经存放在页缓存中，那么这块数据就可以立即返回给应用程序，而不需要经过实际的物理读盘操作。当然，如果数据在应用程序读取之前并未被存放在页缓存中，那么就需要先将数据从磁盘读到页缓存中去。对于写操作来说，应用程序也会将数据先写到页缓存中去，数据是否被立即写到磁盘上去取决于应用程序所采用的写操作机制：如果用户采用的是同步写机制,那么数据会立即被写回到磁盘上，应用程序会一直等到数据被写完为止；如果用户采用的是延迟写机制，那么应用程序就完全不需要等到数据全部被写回到磁盘，数据只要被写到页缓存中去就可以了。在延迟写机制的情况下，操作系统会定期地将放在页缓存中的数据刷到磁盘上。与异步写机制不同的是，延迟写机制在数据完全写到磁盘上的时候不会通知应用程序，而异步写机制在数据完全写到磁盘上的时候是会返回给应用程序的。所以延迟写机制本身是存在数据丢失的风险的，而异步写机制则不会有这方面的担心。

2.缓存I/O的缺点

        在缓存I/O机制中，DMA方式可以将数据直接从磁盘读到页缓存中，或者将数据从页缓存直接写回到磁盘上，而不能直接在应用程序地址空间和磁盘之间进行数据传输，这样的话，数据在传输过程中需要在应用程序地址空间和页缓存之间进行多次数据拷贝操作，这些数据拷贝操作所带来的cpu以及内存开销是非常大的。

二、文件指针

FILE指针：每个被使用的文件都在内存中开辟一个区域，用来存放文件的有关信息，这些信息是保存在一个结构体类型的变量中，该结构体类型是由系统定义的，取名为FILE。

标准I/O库的所有操作都是围绕流(stream)来进行的，在标准I/O中，流用FILE \*来描叙的。

其在Linux操作系统中的定义:  
[http://blog.chinaunix.net/attachment/201205/6/26833883_1336265104PTK7.png](http://blog.chinaunix.net/attachment/201205/6/26833883_1336265104PTK7.png)

三、行缓冲、全缓冲、不缓冲的定义

1.Fully buffered means that I/O takes place only when the buffer is fully, the process explicitly calls fflush, or the process terminates by calling exit. A common size for the standard I/O buffer is 4096 bytes;

2. Line buffered means that I/O takes place when a newline is encountered, when the process calls fflush, or when the process terminates by calling exit.

3. Unbuffered means that I/O take place each time a standard I/O output function is called.

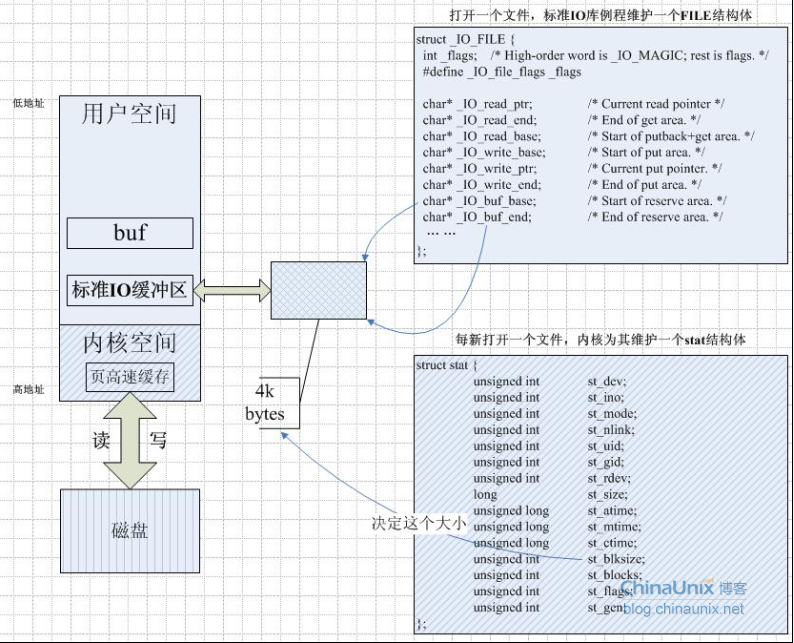
Most unix implementations of the standard I/O libarary use the following rules.

1. Standard error is always unbuffered.

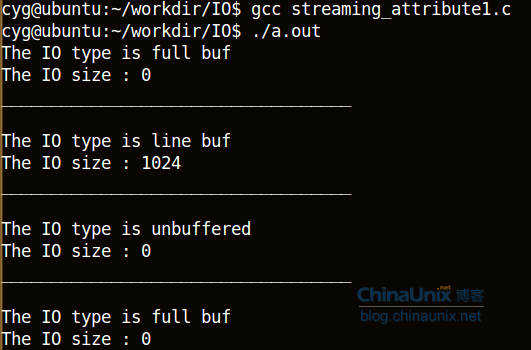
2. Standard input and standard output are fully buffered, unless they refer to a terminal device, in which case, they are line buffered.

3. All other streams are fully buffered unless they refer to a terminal device,

in which case, they are line buffered.

四、一步步探究  
[](http://blog.chinaunix.net/attachment/201205/6/26833883_1336265180N2VX.png)

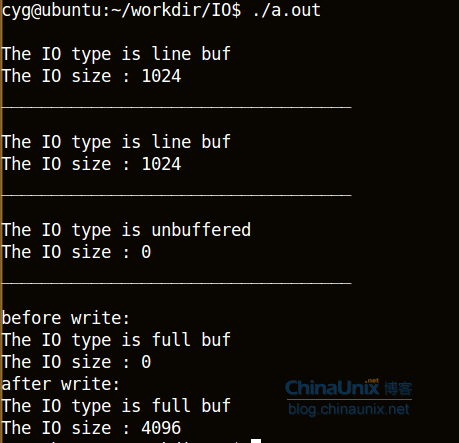
1. #include <stdio.h>
2. int stream\_attribute(FILE \*fp)
3. {
4. if(fp->\_flags & \_IO\_UNBUFFERED)
5. {
6. printf("The IO type is unbuffered\n");
7. }else if(fp->\_flags & \_IO\_LINE\_BUF){
8. printf("The IO type is line buf\n");
9. }else{
10. printf("The IO type is full buf\n");
11. }
12. printf("The IO size : %d\n",fp->\_IO\_buf\_end - fp->\_IO\_buf\_base);
13. return 0;
14. }
15. int main()
16. {
17. FILE \*fp;
18. stream\_attribute(stdin);
19. printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");
20. stream\_attribute(stdout);
21. printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");
22. stream\_attribute(stderr);
23. printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");
24. if((fp = fopen("test.txt","w+")) == NULL)
25. {
26. perror("fail to fopen");
27. }
28. stream\_attribute(fp);
29. return 0;
30. }

运行结果:  
[](http://blog.chinaunix.net/attachment/201205/6/26833883_1336265242O5nI.png)

我们修改一下代码再看

1. #include <stdio.h>
2. int stream\_attribute(FILE \*fp)
3. {
4. if(fp->\_flags & \_IO\_UNBUFFERED)
5. {
6. printf("The IO type is unbuffered\n");
7. }else if(fp->\_flags & \_IO\_LINE\_BUF){
8. printf("The IO type is line buf\n");
9. }else{
10. printf("The IO type is full buf\n");
11. }
12. printf("The IO size : %d\n",fp->\_IO\_buf\_end - fp->\_IO\_buf\_base);
13. return 0;
14. }
15. int main()
16. {
17. FILE \*fp;
18. getchar();
19. stream\_attribute(stdin);
20. printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");
21. stream\_attribute(stdout);
22. printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");
23. stream\_attribute(stderr);
24. printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");
25. if((fp = fopen("test.txt","w+")) == NULL)
26. {
27. perror("fail to fopen");
28. }
29. printf("before write:\n");
30. stream\_attribute(fp);
31. fputc('a',fp);
32. printf("after write:\n");
33. stream\_attribute(fp);
34. return 0;
35. }

运行结果:

[](http://blog.chinaunix.net/attachment/201205/6/26833883_1336265378694T.png)  
从以上我们可以看出，标准I/O并不是一开始就分配的，只有进行了输入或者输出的操作才进行分配的，标准输入和标准输出默认是全缓冲，但是如果和终端关了就是行缓冲。可以看到在linux操作系统中行缓冲的大小是1k，全缓冲的大小是4k。