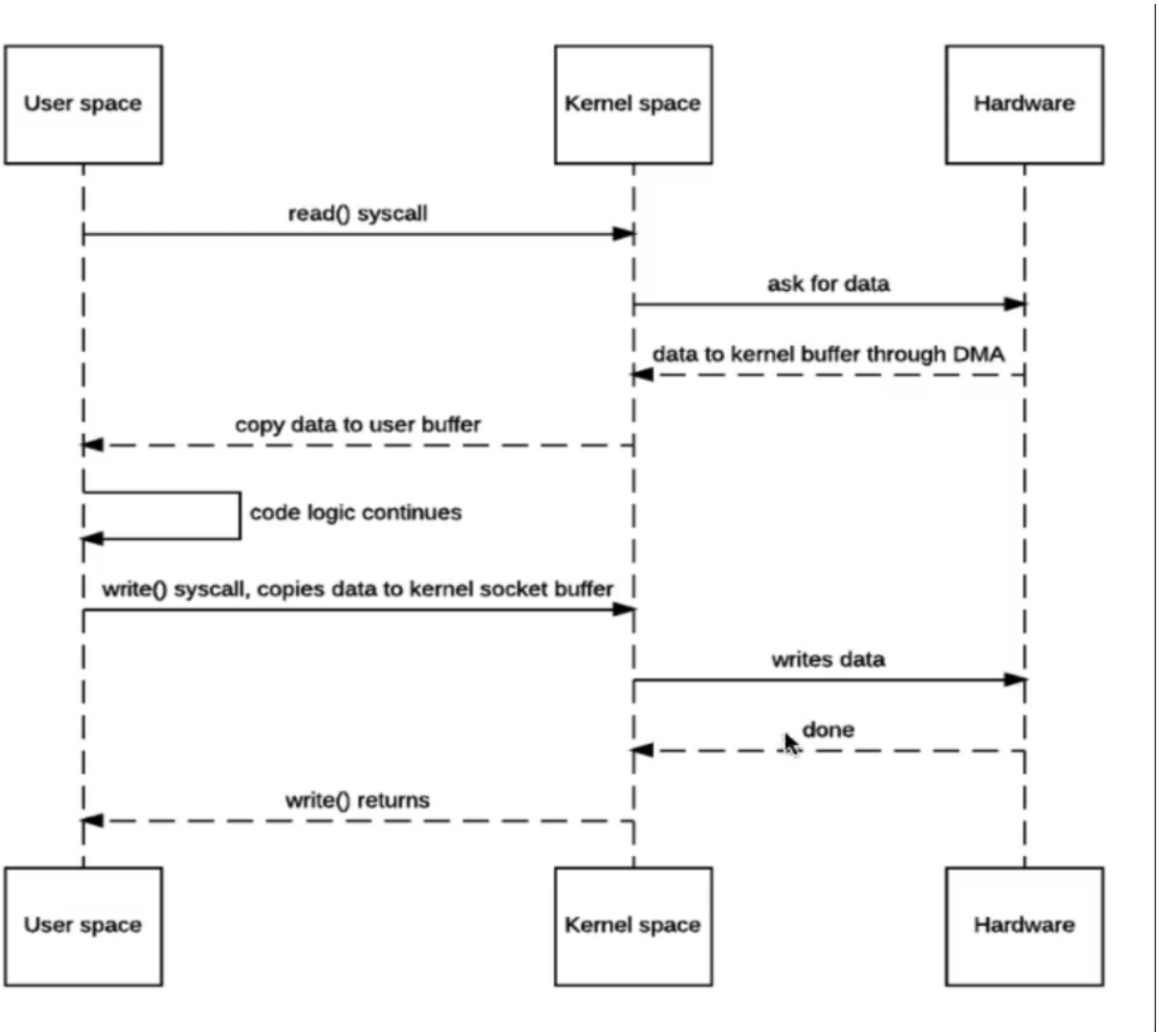
1.传统操作拷贝：



User space：用户空间

Kemel space：内核空间

Hardware：磁盘空间

read：

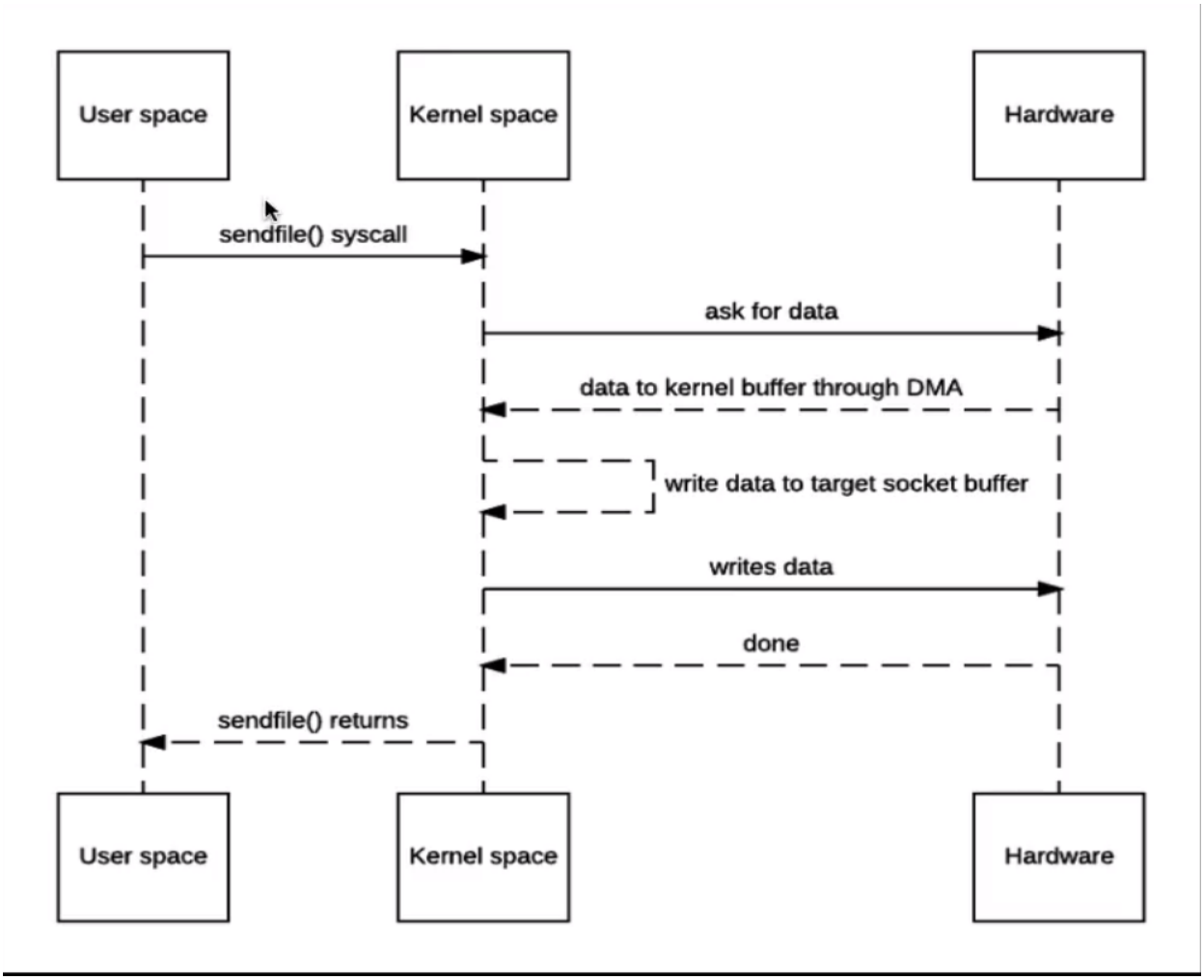
用户空间通过调用native方法调用底层库方法（根据操作系统，大多都是c文件），从用户空间切换到内核空间（**上下文切换一次**），调用操作系统的read(),内核空间从磁盘copy数据到内核缓冲区内存空间（**拷贝一次数据**），再将内核空间缓冲区的数据copy到用户空间缓冲区（**拷贝一次数据**），内核空间也需要切换到用户空间（**上下文切换一次**），一次read操作执行了**两次上下文切换，两次拷贝数据**。

write：

用户空间缓冲区copy到内核空间socket缓冲区（**上下文切换一次**），用户空间切换到内核空间（**拷贝一次数据**），内核空间写数据到磁盘空间（**拷贝一次数据**），内核空间切换到用户空间（**上下文切换一次**），一次write操作执行了**两次上下文切换，两次拷贝数据**。

一次读写操作执行了**四次次上下文切换，四次拷贝数据**。

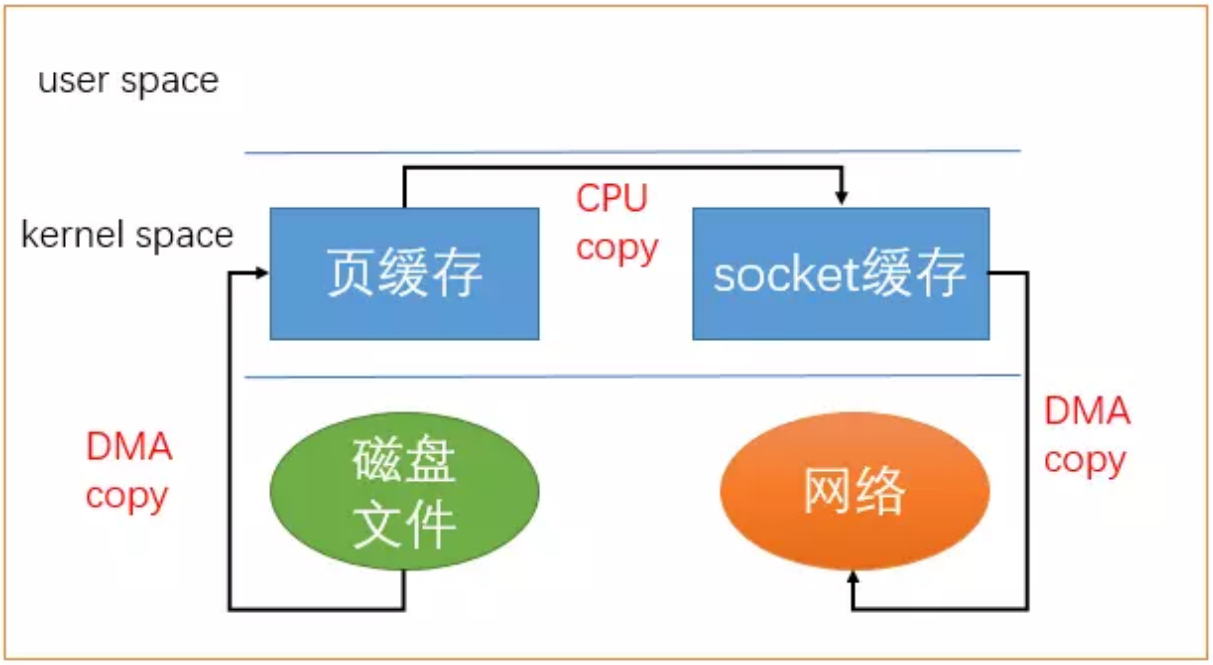
2.零拷贝（操作系统意义上）



操作系统意义上的零拷贝，没有内核空间copy到用户空间

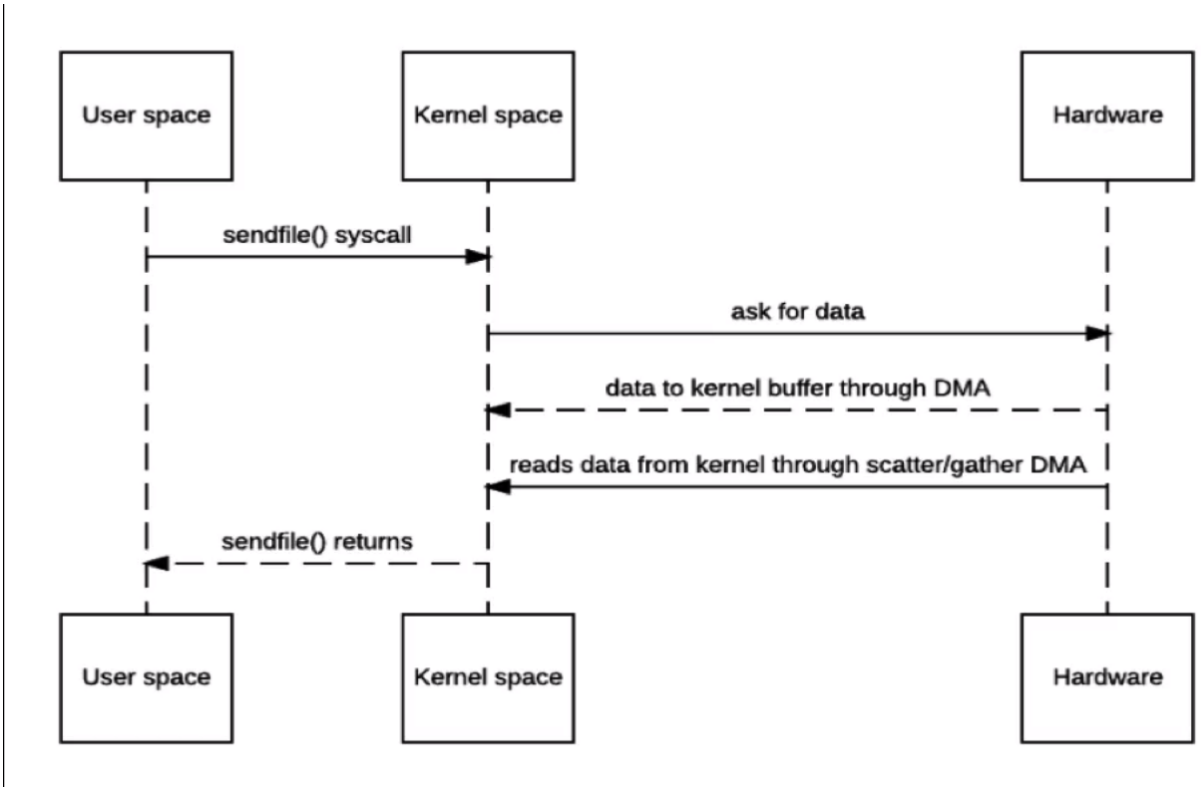
所有的操作都是在内核空间中进行操作，没有内核和用户之间的拷贝过程

但是在内核空间中有拷贝（内核缓冲区拷贝到socket缓冲区）过程



此时数据进行了三次拷贝，两次是内核空间到磁盘文件，一次是内核空间到socket缓存

3.零拷贝（直接操作磁盘空间）

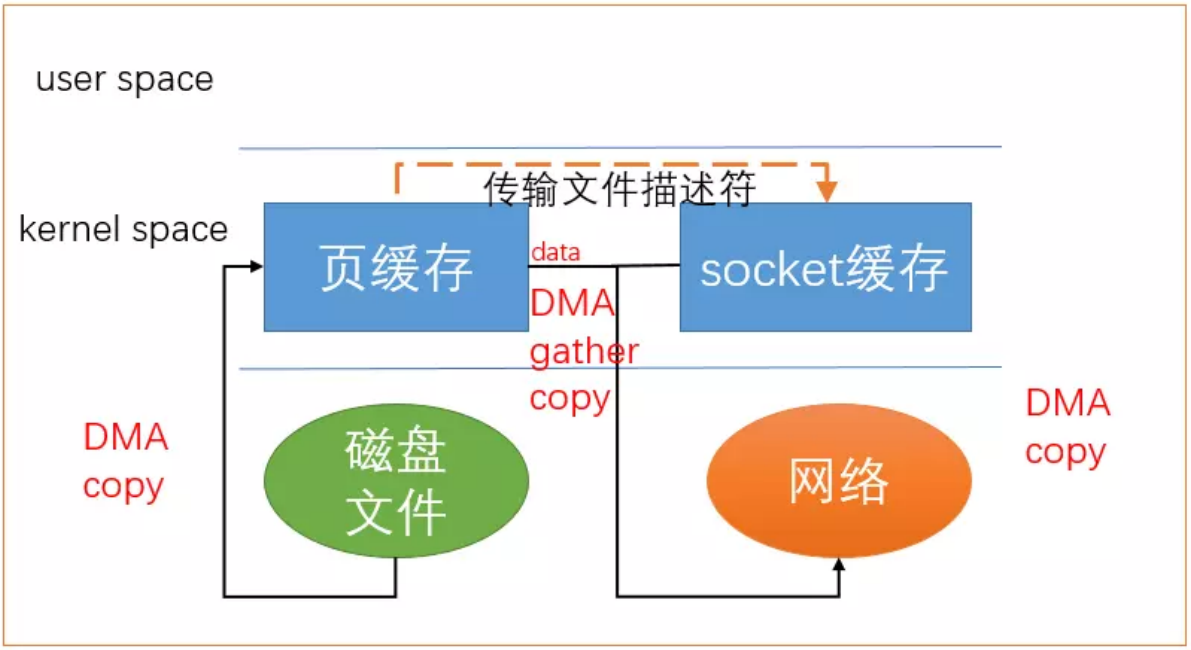


不需要从内核缓冲区拷贝到socket缓冲区

用户只能发送指令，无法参与修改文件。

此时就需要通过内存映射文件加以实现

MappedByteBuffer



此时不需要copy文件到socket缓存，只需要将文件描述符传输到socket缓存（**文件长度，文件在内存空间地址**），此时只需要拷贝2次，磁盘到内核，内核到磁盘，真正实现了零拷贝，用户空间通过socket缓存的文件描述符操作文件。

比较传统io和新io操作效率比较

传统服务端

```
1  public class OldIOServer {
2   public static void main(String[] args) throws Exception{
3   ServerSocket serverSocket = new ServerSocket(8899);
4
5   while (true){
6   Socket socket = serverSocket.accept();
7
8   DataInputStream dataInputStream = new DataInputStream(socket.ge
tInputStream());
9
10   try {
11   byte[] byteArray = new byte[4096];
12   long totalCount = 0;
13   while (true){
14   int readCount = dataInputStream.read(byteArray, 0, byteArray.l
ength);
```

```
15   totalCount += readCount;
16   System.out.println("当次读取" + readCount + "字节...");
17   System.out.println("总共读取" + totalCount + "字节...");
18   }
19   }catch (Exception e){
20   e.printStackTrace();
21   }
22   }
23   }
24  }
```

传统客户端

```
1  public class OldIOClient {
2   public static void main(String[] args) throws Exception{
3
4
5   String filePath = "D:\\BaiduNetdiskDownload\\cn_visio_2010_x64_
 516562.exe";//测试用文件大约500MB
6
7   try (Socket client = new Socket("localhost", 8899);
8   InputStream inputStream = new FileInputStream(filePath);
9   DataOutputStream dataOutputStream = new
 DataOutputStream(client.getOutputStream()))
10   {
11
12   byte[] buffer = new byte[4096];
13   long readCount;
14   long totalCount = 0;
15
16   long startTime = System.currentTimeMillis();
17
18   while ((readCount = inputStream.read(buffer)) >= 0){
19   totalCount += readCount;
20   dataOutputStream.write(buffer);
21   }
22   System.out.println("发送总字节数" + totalCount + ", 耗时: " + (S
 ystem.currentTimeMillis() - startTime));
```

```
23
24
25   }catch (Exception e){
26   e.printStackTrace();
27   }
28   }
29  }
```

新IO服务端

```
1  public class NewIOServer {
2   public static void main(String[] args) throws Exception {
3   InetSocketAddress address = new InetSocketAddress(8899);
4
5   ServerSocketChannel serverSocketChannel = ServerSocketChannel.o
pen();
6   ServerSocket serverSocket = serverSocketChannel.socket();
7   serverSocket.setReuseAddress(true);
8   serverSocket.bind(address);
9
10   ByteBuffer byteBuffer = ByteBuffer.allocate(4096);
11
12   while (true){
13   SocketChannel socketChannel = serverSocketChannel.accept();
14   socketChannel.configureBlocking(true);
15
16   int readCount = 0;
17   while (readCount != -1){
18   try {
19   readCount = socketChannel.read(byteBuffer);
20   }catch (Exception e){
21   e.printStackTrace();
22   }
23   byteBuffer.rewind();
24   }
25
26   }
```

```
    }
}
```

新IO客户端

```
public class NewIOClient {
    public static void main(String[] args) throws Exception{
        InetSocketAddress address = new InetSocketAddress("localhost", 8899);
        SocketChannel socketChannel = SocketChannel.open();
        socketChannel.connect(address);
        socketChannel.configureBlocking(true);

        String filePath = "D:\\BaiduNetdiskDownload\\cn_visio_2010_x64_516562.exe";//测试用文件大约500MB
        FileChannel fileChannel = new FileInputStream(filePath).getChannel();

        long startTime = System.currentTimeMillis();
        //从0开始，写入整个长度，写到socketChannel
        /**
        * 会把长度设置为2147483647L也就是大概2GB的大小
        * 所以需要对FileChannel.size()返回值进行判断,当它返回值大于0时始终要执行transferTo方法
        * 因为transferTo单次只能处理2gb左右的长度,同时计算position偏移量
        */
        fileChannel.transferTo(0, fileChannel.size(), socketChannel);
        System.out.println("发送总字节数" + fileChannel.size() + ", 耗时: " + (System.currentTimeMillis() - startTime));
        fileChannel.close();
    }
}
```

比较结果

需要注意点

transferTo()

方法需要注意**单次最大传输2G**即2147483647L

根据源码：

```
1 public long transferTo(long var1, long var3, WritableByteChannel
var5) throws IOException {
2  this.ensureOpen();
3  if (!var5.isOpen()) {
4  throw new ClosedChannelException();
5  } else if (!this.readable) {
6  throw new NonReadableChannelException();
7  } else if (var5 instanceof FileChannelImpl && !((FileChannelImp
l)var5).writable) {
8  throw new NonWritableChannelException();
9  } else if (var1 >= 0L && var3 >= 0L) {
10  long var6 = this.size();
11  if (var1 > var6) {
12  return 0L;
13  } else {
14  int var8 = (int)Math.min(var3, 2147483647L);
15  if (var6 - var1 < (long)var8) {
```

```
  var8 = (int)(var6 - var1);
  }

  long var9;
  if ((var9 = this.transferToDirectly(var1, var8, var5)) >= 0L)
{
  return var9;
  } else {
  return (var9 = this.transferToTrustedChannel(var1, (long)var8,
var5)) >= 0L ? var9 : this.transferToArbitraryChannel(var1, var8,
var5);
  }
  }
  } else {
  throw new IllegalArgumentException();
  }
 }
```