链滴

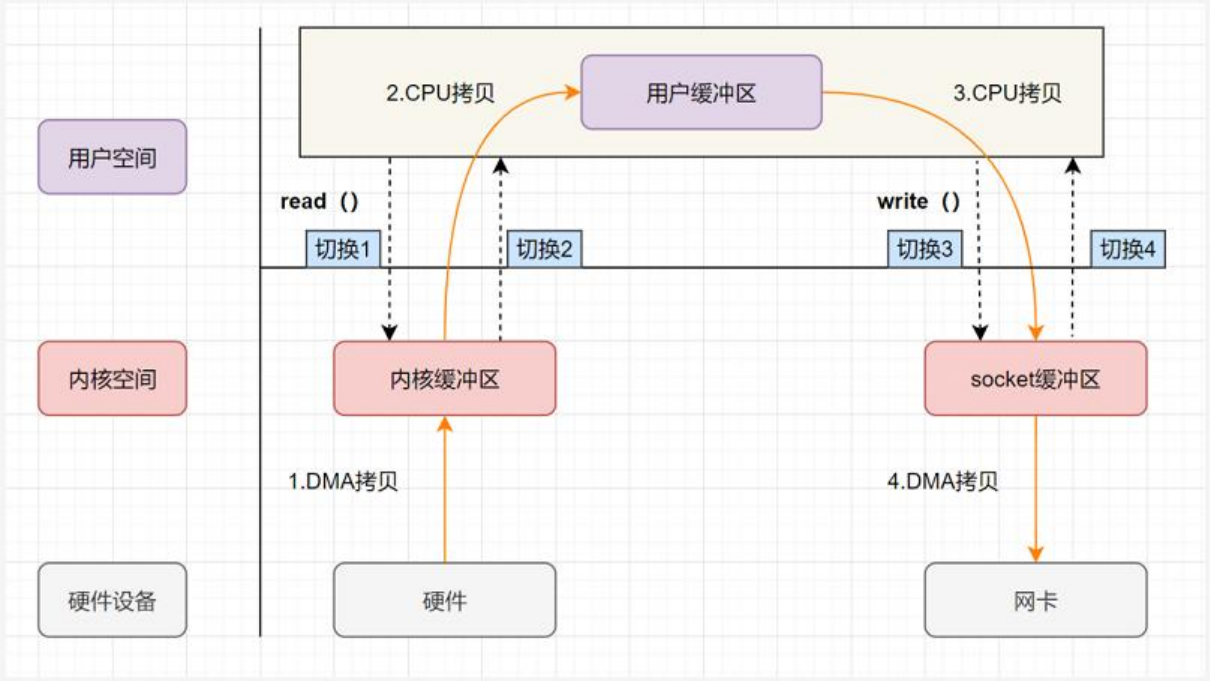
IO 零拷贝

作者：AshShawn

原文链接：https://ld246.com/article/1650383793071

来源网站：链滴

1.传统IO拷贝



如图所示,发生4次上下文切换以及4次数据拷贝(2次CPU拷贝以及2次DMA拷贝)

2.零拷贝的几种方式

- mmap+write
- sendfile
- 带有DMA收集拷贝功能的sendfile

2.1mmap+write

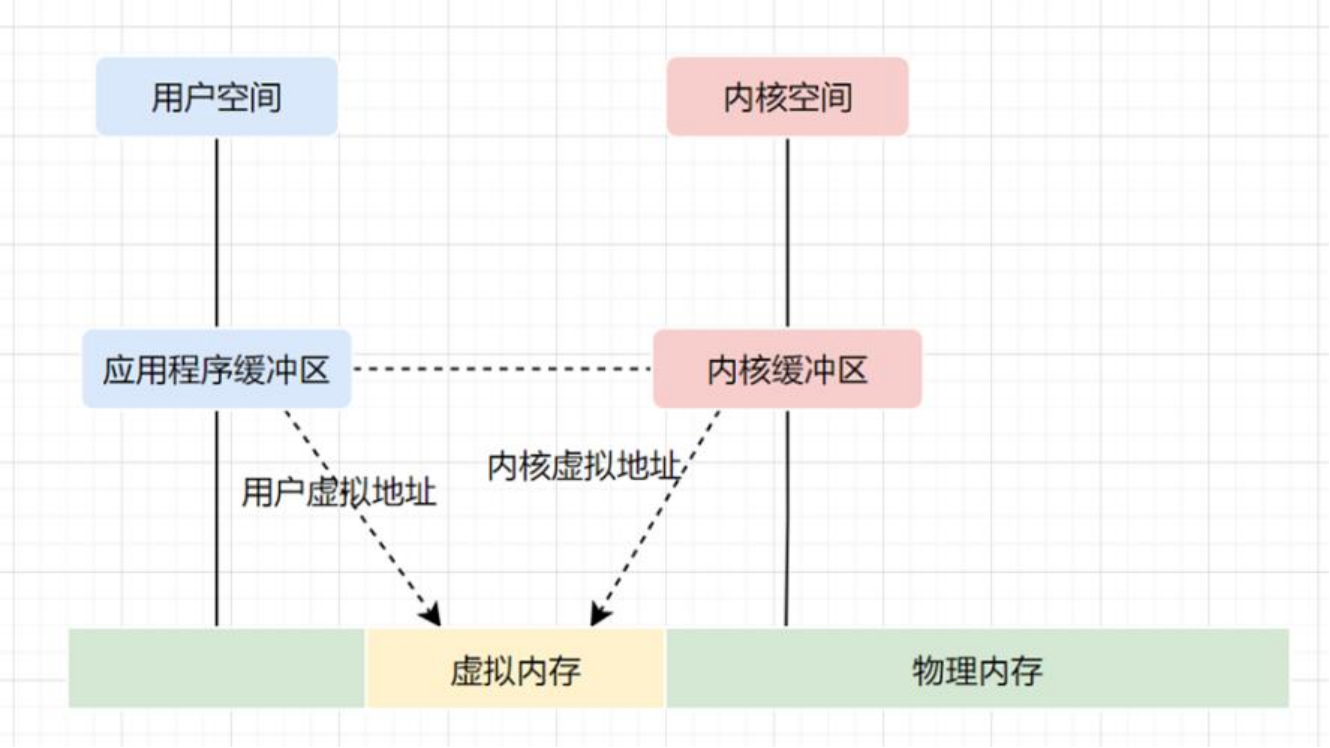
虚拟内存

现代操作系统使用虚拟内存，即虚拟地址取代物理地址，使用虚拟内存可以有2个好处：

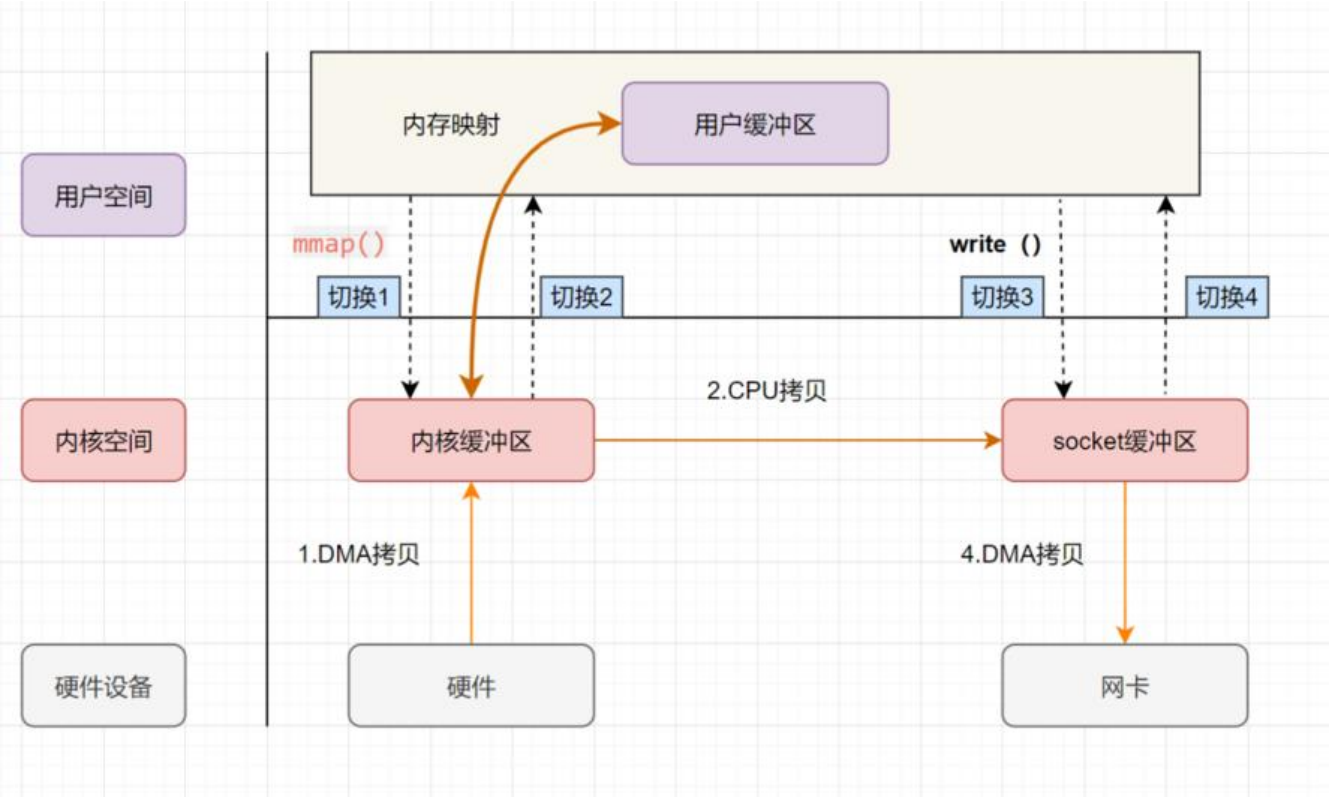
- 虚拟内存空间可以远远大于物理内存空间
- 多个虚拟内存可以指向同一个物理地址

正是**多个虚拟内存可以指向同一个物理地址**，可以把内核空间和用户空间的虚拟地址映射到同一个物地址，这样的话，就可以减少IO的数据拷贝次数啦，示意图如下

原文链接：IO 零拷贝



mmap



- 用户进程通过 mmap方法向操作系统内核发起IO调用，**上下文从用户态切换为内核态** 。
- CPU利用DMA控制器，把数据从硬盘中拷贝到内核缓冲区。
- **上下文从内核态切换回用户态** ， mmap方法返回。
- 用户进程通过 write方法向操作系统内核发起IO调用，**上下文从用户态切换为内核态** 。

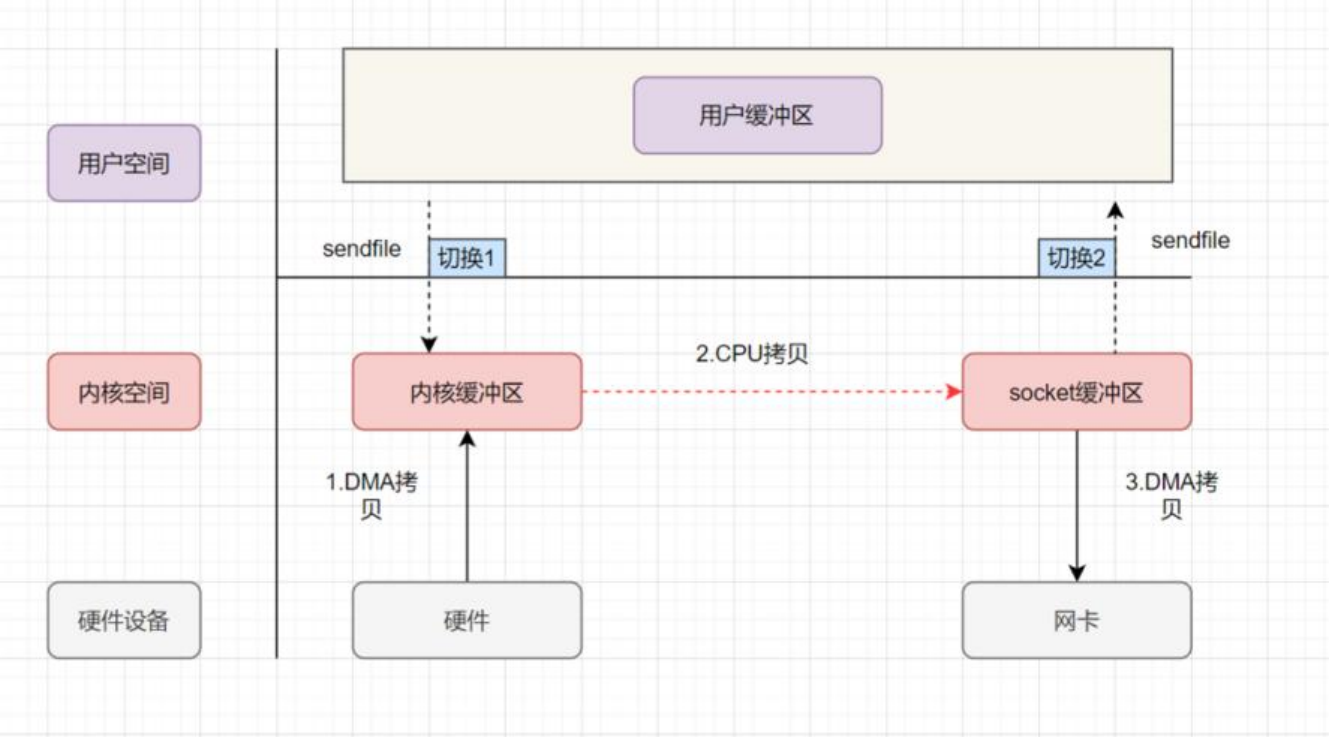
原文链接：IO 零拷贝

- CPU将内核缓冲区的数据拷贝到的socket缓冲区。
- CPU利用DMA控制器，把数据从socket缓冲区拷贝到网卡， **上下文从内核态切换回用户态** ，writ调用返回。

可以发现，mmap+write实现的零拷贝，I/O发生了**4** 次用户空间与内核空间的上下文切换，以及3次据拷贝。其中3次数据拷贝中，包括了**2次DMA拷贝和1次CPU拷贝** 。

mmap是将读缓冲区的地址和用户缓冲区的地址进行映射，内核缓冲区和应用缓冲区共享，所以节省一次CPU拷贝 ‘’ 并且用户进程内存是**虚拟的** ，只是**映射** 到内核的读缓冲区，可以节省一半的内存间。

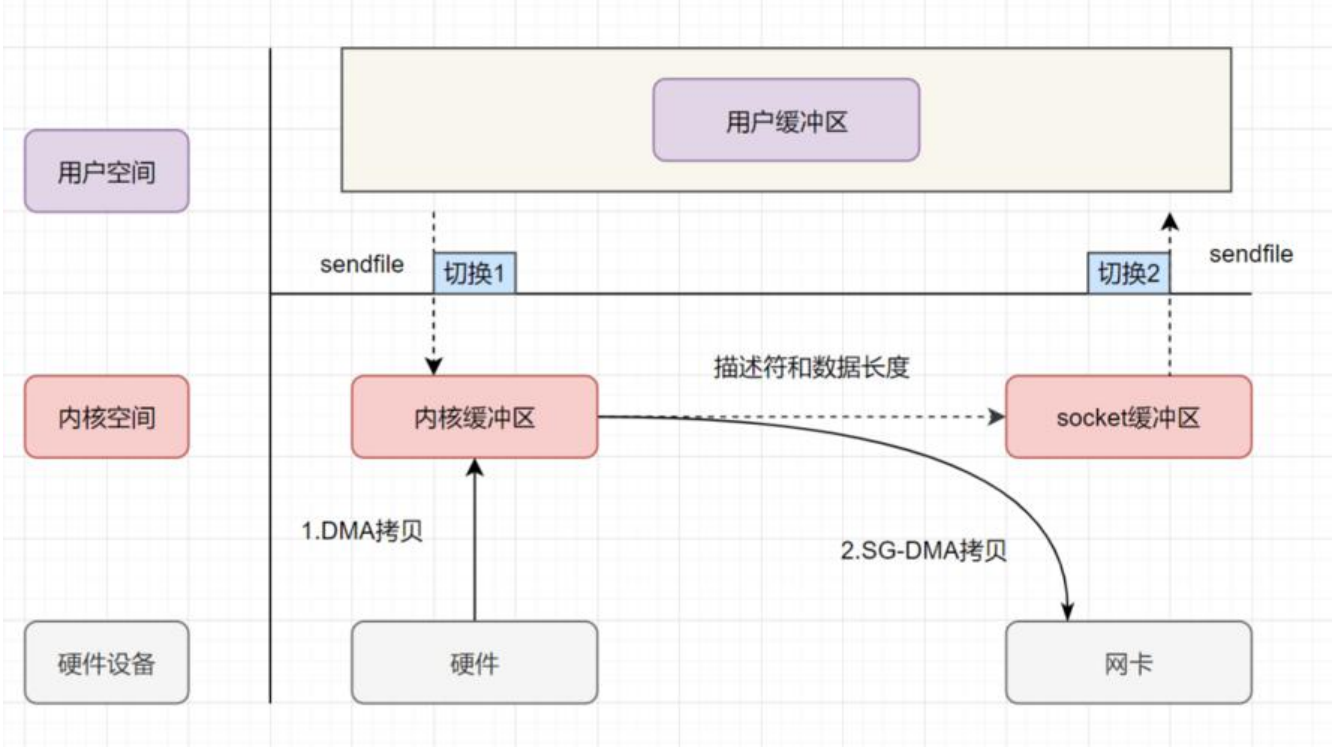
2.2 sendfile



1. 用户进程发起sendfile系统调用， **上下文（切换1）从用户态转向内核态**
2. DMA控制器，把数据从硬盘中拷贝到内核缓冲区。
3. CPU将读缓冲区中数据拷贝到socket缓冲区
4. DMA控制器，异步把数据从socket缓冲区拷贝到网卡，
5. **上下文（切换2）从内核态切换回用户态** ，sendfile调用返回

sendfile实现的零拷贝，I/O发生了**2** 次用户空间与内核空间的上下文切换，以及3次数据拷贝。其中3数据拷贝中，包括了**2次DMA拷贝和1次CPU拷贝** 。那能不能把CPU拷贝的次数减少到0次呢？有的即带有DMA收集拷贝功能的sendfile！

2.3 sendfile+DMA scatter/gather



1. 用户进程发起sendfile系统调用， **上下文（切换1）从用户态转向内核态**

2. DMA控制器，把数据从硬盘中拷贝到内核缓冲区。

3. CPU把内核缓冲区中的 **文件描述符信息** （包括内核缓冲区的内存地址和偏移量）发送到socket缓冲区

4. DMA控制器根据文件描述符信息，直接把数据从内核缓冲区拷贝到网卡

5. **上下文（切换2）从内核态切换回用户态** ，sendfile调用返回。

可以发现，sendfile+DMA scatter/gather实现的零拷贝，I/O发生了**2** 次用户空间与内核空间的上文切换，以及2次数据拷贝。其中2次数据拷贝都是包**DMA拷贝** 。这就是真正的 **零拷贝（Zero-copy)** 技术，全程都没有通过CPU来搬运数据，所有的数据都是通过DMA来进行传输的。

3. Java 中的零拷贝

3.1 mmap

Java NIO有一个MappedByteBuffer的类，可以用来实现内存映射

```
try {
        //读取文件
            FileChannel readChannel = FileChannel.open(Paths.get("./jay.txt"), StandardOpenOpti
n.READ);
        //mmap映射读取文件
            MappedByteBuffer data = readChannel.map(FileChannel.MapMode.READ_ONLY, 0, 10
4 * 1024 * 40);
        //获取写入通道
            FileChannel writeChannel = FileChannel.open(Paths.get("./siting.txt"), StandardOpenO
tion.WRITE, StandardOpenOption.CREATE);
            //数据传输,从mappedBytebuffer中写入到目标通道
            writeChannel.write(data);
```

```
            readChannel.close();
            writeChannel.close();
        }catch (Exception e){
            System.out.println(e.getMessage());
        }
```

3.2 sendfile

FileChannel的transferTo()/transferFrom()，底层就是sendfile() 系统调用函数。Kafka 这个开源项就用到它

```
try {
            FileChannel readChannel = FileChannel.open(Paths.get("./jay.txt"), StandardOpenOpti
n.READ);
            long len = readChannel.size();
            long position = readChannel.position();

            FileChannel writeChannel = FileChannel.open(Paths.get("./siting.txt"), StandardOpenO
tion.WRITE, StandardOpenOption.CREATE);
            //数据传输
            readChannel.transferTo(position, len, writeChannel);
            readChannel.close();
            writeChannel.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
```

4.直接内存

```
package com.sq.oom;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.util.Date;

import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
@Slf4j
public class OomApplication {

    public static void main(String[] args) {
        SpringApplication.run(OomApplication.class, args);
    }
```

```
    @RequestMapping("/test")
    public Object testOom() throws FileNotFoundException {

        //启动参数 -XX:MaxDirectMemorySize=2m -verbose:gc -XX:+PrintGCDetails  限制了最大
接内存为2MB
        //第一次调用没有任何问题,第二次调用直接OOM
        service();
        return "success";
    }

    public void service() throws FileNotFoundException {
        String path = System.getProperty("java.io.tmpdir") + File.separator;
        log.info("file-path={}", path);
        String fileName = "test.DAT";
        String fName = path + fileName;

        File file = new File(fName);
        if (file.exists()) {
            file.delete();
        }
        RandomAccessFile rcf = new RandomAccessFile(fName, "rw");
        FileChannel channel = rcf.getChannel();
        int outLoopSize = 10;
        int innerLoopSize = 2000;
        long offset = 0;

        ByteBuffer byteBuffer = ByteBuffer.allocateDirect(2 * 1024 * 1024);

        try {
            for (int i = 0; i < outLoopSize; i++) {
                StringBuilder builder = new StringBuilder();
                for (int j = 0; j < innerLoopSize; j++) {

                    builder.append(i+"39xxxxxxxx");
                }
                byte[] bytes = builder.toString().getBytes();
                System.out.println("bytes大小" + bytes.length);
                byteBuffer.put(bytes, 0, bytes.length);
                byteBuffer.flip();
                while (byteBuffer.hasRemaining()) {
                    channel.write(byteBuffer);
                }
                channel.force(true);
                byteBuffer.clear();
                offset += bytes.length;
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                channel.close();
                rcf.close();
            } catch (IOException e) {
```

```
                e.printStackTrace();
            }
        }
    }
}
```