0716221 余忠旻

Section 1: Analysis of Baseline

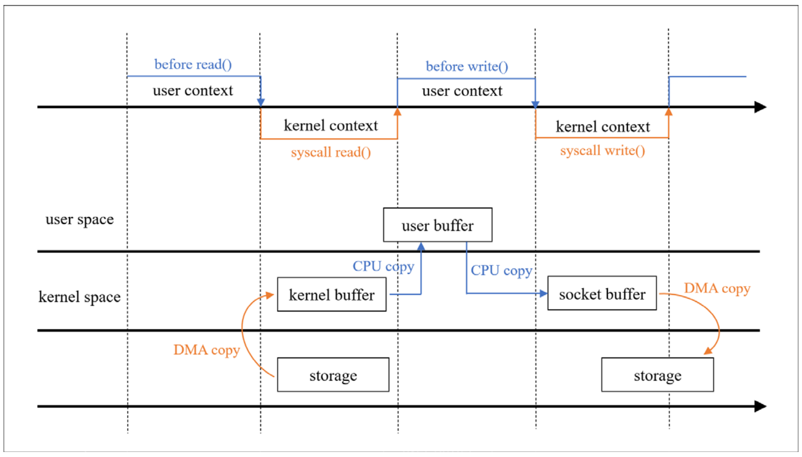
Baseline program uses read() and write() function to achieve duplicating a file.

For read(), it involves these steps:

1. Switching from the user mode to the kernel mode.
2. Copying data from the storage to kernel buffer.
3. Copying data from the kernel buffer to user buffer.
4. Switching from the kernel mode to the user mode.

For write(), it involves these steps:

1. Switching from the user mode to the kernel mode.
2. Copying data from the user buffer to socket buffer.
3. Copying data from the socket buffer to storage.
4. Switching from the kernel mode to the user mode.



Above these steps can show that we need four times of data copying and four times of switching. This is the bottleneck of the performance. We can reduce the times of data copying and switching to achieve better performance.

Section 2: Proposed method

I think we can reduce the number of times about data copying and switching. In other words, we can use **zero-copy mechanism** to improve performance for duplicating a file. So, I found three methods that use zero-copy mechanism to improve the performance. The underlying transmission process of three methods is similar. They are **sendfile()**, **splice()**, **copy_file_range()**.

Section 3: Performance evaluation

➢ Test platform:

CPU: Intel Core i7-10870H

Ubuntu 20.04 LTS (VM): Memory 4GB, Uniprocessor

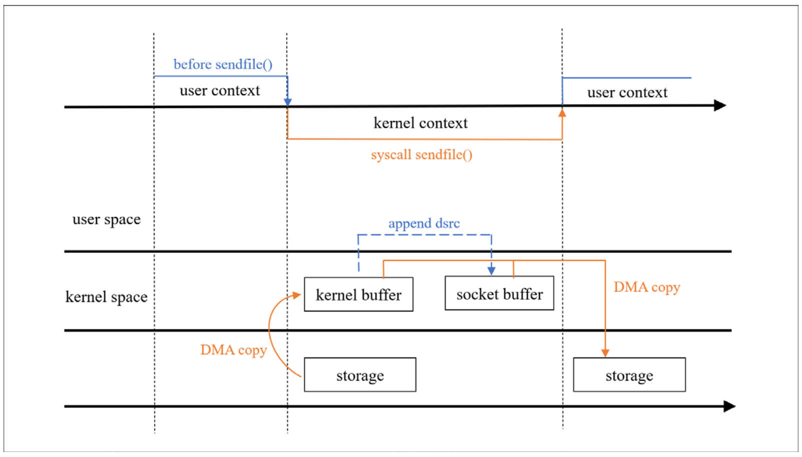
➢ Present the measured performance results of baseline and my three methods.

	Baseline	sendfile()	splice()	copy_file_range()
Time (s)	0.814	0.288	0.291	0.289
Time	100%	35%	35%	35%

The sendfile() function in Linux kernels before 2.6.33, its out_fd must refer to a socket. Since Linux 2.6.33 it can be any file. So, we can use sendfile() to duplicate a file.

In sendfile() function, we can see the below graph to know its mechanism.

1. Switching from the user mode to the kernel mode.
2. Copying data from the storage to kernel buffer.
3. Append the data position and offset in the kernel buffer to the socket buffer.
4. According to data position and offset information in socket buffer, copy data to storage.
5. Switching from the kernel mode to the user mode.



Above these steps can show that we need two times of data copying and two times of switching. These two times of data copy is DMA copy. Compared to baseline program, we need to four times of data copying and four times of switching. These data copying include two time of CPU copy and two times of DMA copy. So, my program can **avoid the two times of CPU copy** and **reduce two times of switching**, which improves the performance a lot.

In splice(), it moves data between two file descriptors without copying between kernel address space and user address space. Its transmission mechanism is similar with sendfile()'s. A little difference is that splice() has to set up the pipe between kernel buffer and socket buffer, which sendfile() doesn't need.

In copy_file_range(), I looked up the man page of Linux. It performs an in-kernel copy between two file descriptors. Its transmission mechanism is similar with sendfile()'s or splice()'s. The man page mentions that copy_file_range() system call first appeared in Linux 4.5. However, it is a nonstandard Linux and GNU extension. It maybe not applicable to some environment.

References

1. https://zh.wikipedia.org/zh-tw/%E9%9B%B6%E5%A4%8D%E5%88%B6
2. https://www.gushiciku.cn/pl/2wMB/zh-tw
3. https://hackmd.io/@sysprog/linux2020-zerocopy#sendfile-%E7%B3%BB%E7%B5%B1%E5%91%BC%E5%8F%AB
4. https://man7.org/linux/man-pages/man2/sendfile.2.html
5. https://man7.org/linux/man-pages/man2/splice.2.html
6. https://man7.org/linux/man-pages/man2/copy_file_range.2.html