Operating System Programming Assignment: Fast File Duplication

Name: 周俊川
Student ID: 310551002

Abstract

In the “Fast File Duplication” assignment, I proposed using `*sendfile*` system call instead of `*read*` + `*write*` system calls (baseline method) for large file duplication. It shows that using `*sendfile*` can achieve around 4.5 times faster compared to the baseline method.

Analysis of Baseline

In this assignment, we will duplicate a huge file (1GB) on tmpfs to ignore the storage latency, in other words, the data we want to duplicate was in RAM (kernel page) already. The file operations workflow of baseline method is as below:

1. Define cache buffer size: 1024 bytes;
2. Open source file and destination file;
3. Read a portion (1024 bytes) of data from source kernel page to user buffer;
4. Write the data to destination kernel page;
5. Repeat (3) and (4) until no any data to read;
6. Close source file and destination file.

The workflow mentioned above has some overheads:

I. In step 5, it requires a loop to read and write a piece of data repeatedly, which makes large context switches overhead. Since the read data will be allocated in the stack area, it is hard to read a large file of data at once because the memory of the stack area can be allocated very little.
   One thing we can do is try to tune the cache buffer size to optimize the execution time.
   In addition, instead of tuning the cache buffer size manually, we can also allocate the content into the heap area to access the data at once.

II. In step 3 and 4, `*read*` and `*write*` system calls require data movement between user space and kernel space. While accessing data in the user space is necessary for the `*read*` and `*write*` method, it also produces extra overhead on it.
   At the same time, each of the system call (`*read*` and `*write*`) will switch from the user mode to kernel mode when executing the system call, and switch from the kernel mode to user mode after finish, so the `*read*` + `*write*` system calls occurs 4 context switch in one round.

Proposed method

Although the first overhead could be solved by several methods, the second overhead mentioned above is hard to resolve when using `*read*` + `*write*` system calls. Here I proposed a method: using **zero-copy** technique (`*sendfile*`), to duplicate large files.

Zero-copy technique is often used in copying / transfering a file to another host, meaning from disk file to network. The purpose of zero-copy technique is to eliminate the

unnecessary copies between kernel space and user space. With the implementation of `*sendfile*`, it claims to make data transfer happening in the kernel space only. At the same time, it also requires less context switches than `*read*` + `*write*` system calls.

Performance evaluation

In addition to proposed method, I also compare with the below alternative methods which mentioned in the first overhead,

1. Change the cache buffer size to 1000000 bytes;
2. Allocate in heap area: `*char *buf = new char[size]*`.

I used *perf* to get the analysis result. The result is show as below

	baseline	alternative 1	alternative 2	proposed sol.
exec. time (secs)	1.687	0.392	0.422	**0.359**
user time (secs)	0.301	0.000	0.017	**0.000**
sys. time (secs)	1.375	0.390	0.400	**0.358**
context switches	114	29	29	**19**
page faults	45	289	116	46

According to the result shown above, the proposed solution gets the best execution time over the other methods. We can find that the alternative 1 also gets a large improvement, but it requires manually tuning effort, which might not be suitable for the other situations.

Besides, the context switches of the proposed solution is also the lowest over the other methods because it eliminates the unnecessary mode switches and data copies between kernel space and user space.

Although alternative 1 and alternative 2 also achieve a great improvement in terms of execution time, their page faults are higher than the proposed solution. This reflects that accessing a large portion of content in user space is not a good sign since it will possibly trigger large portions of page faults.

Summary

In this assignment, I showed that using `sendfile` could get a better execution time (4.5 times faster) than the baseline method. Although I showed the other 2 alternative methods also achieve great improvement, it turned out that their context switches and page faults are larger than `*sendfile*`, hence `*sendfile*` solution should be the best choice for duplicating large files instead of `*read*` + `*write*` combination.

References

[1] https://medium.com/swlh/linux-zero-copy-using-sendfile-75d2eb56b39b
[2] http://wiki.csie.ncku.edu.tw/embedded/perf-tutorial#perf-list