OS 2019 Project 2 Report

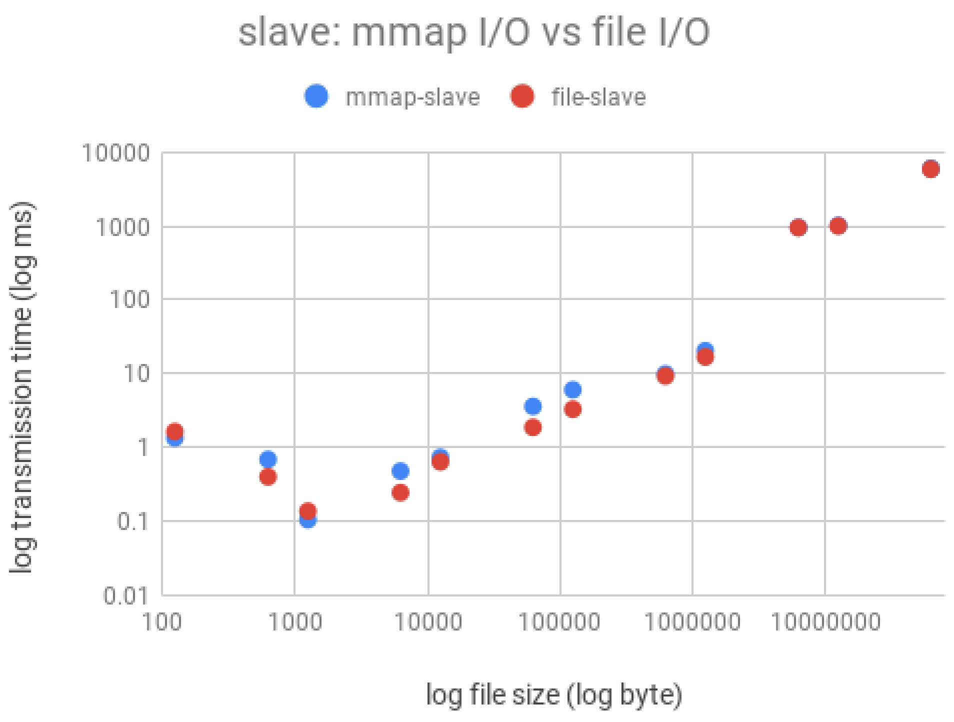
Programming design

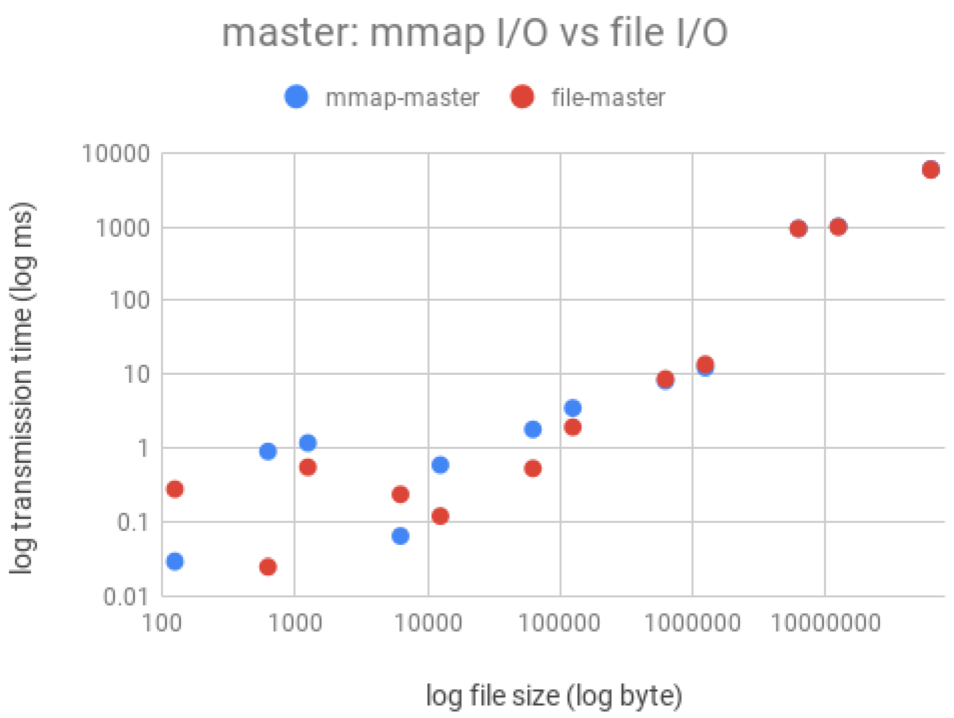
The goal of this project is to compare the performance of file I/O and mmap I/O under different scenarios and explain its superiority. We implement mmap I/O by passing the file descriptor of our target file and its size into mmap as the initial address and mapping size, respectively, and directly rely on the virtual address to access the target file. The following sections explains the implementation of master and slave for mmap I/O:

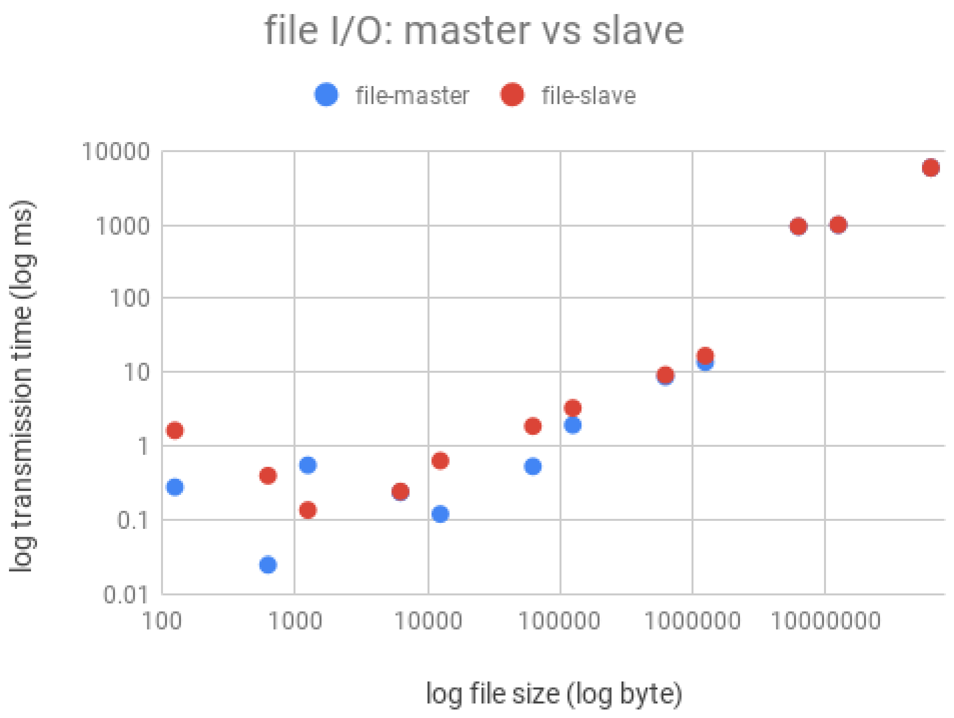
- Master: We directly map the file onto the virtual memory address since we have the file size and its file descriptor beforehand, thus avoiding the cost of intial copying of the file to memory. However, there is no free lunch as we have to construct the mmap which takes time as well.
- Slave: Since the slave does not know the file size, we allocate memory with fixed size initially, conveniently multiples of page, and allocate a new page until the file is fully recieved. One technical difficulty we face here is that directly mapping an empty file to a new memory address will cause bus error. Thus, we have to write dummy character at the corresponding offset position in the file and remove it after the file is fully sent.

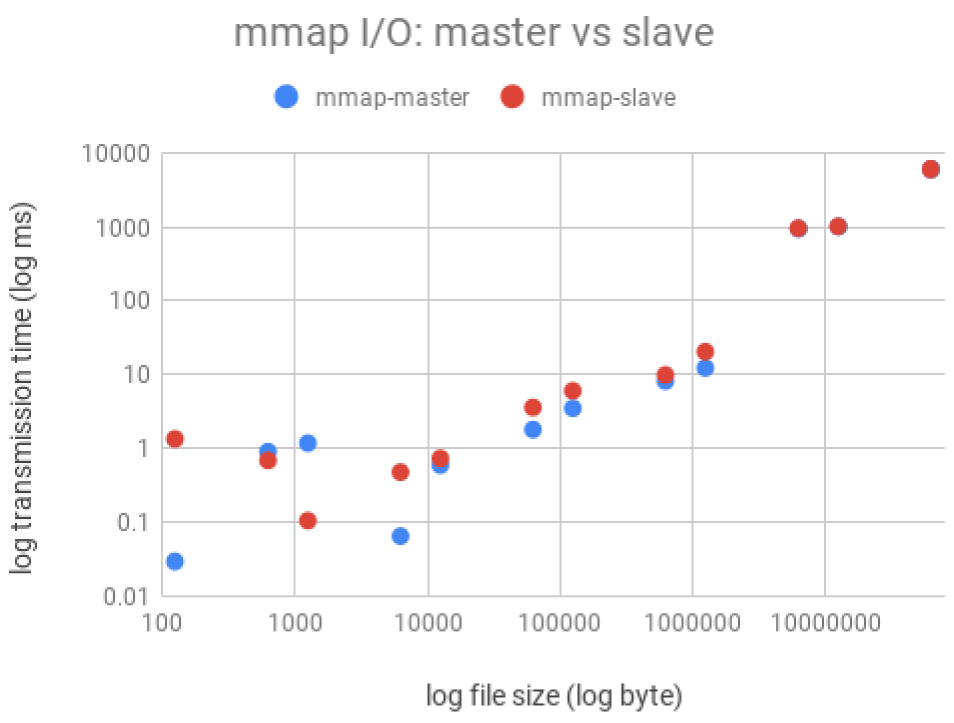
Experiment Result

The following graph contains comparisons of fixed combinations of master and slave product with file I/O and mmap I/O. All of the data is taken logged scale due to the large variance in experiment data range to avoid stacking of data points for smaller values.




master: mmap I/O vs file I/O
mmap-master
file-master
log transmission time (log ms)
10000
1000
100
10
1
0.1
0.01
100
1000
10000
100000
1000000
10000000
log file size (log byte)


file I/O: master vs slave
file-master
file-slave
log transmission time (log ms)
10000
1000
100
10
1
0.1
0.01
100
1000
10000
100000
1000000
10000000
log file size (log byte)



Comparison between file I/O and mmap I/O

Since file I/O implemented natively by the operation system will check for the file cache in memory first, then copy the file to memory if cache miss, generally if we ignore the cost of constructing mmap, mmap I/O will be faster than file I/O since mmap I/O skips the copying step by directly mapping position of file on disk to memory address. However, since mmap requires time because we have to modify the page tables with multiple system calls, the previous assumption does not hold in reality. Where mmap I/O shines is when we random access the file or re-read the file, since for file I/O these types of operations require copying of the segment of the file into memory again. On the other hand, for mmap I/O, initializing the mmap is a one time cost so it will be amortized over multiple file operations because we can directly go to the position via accessing memory (significantly faster than copying to memory). We will now analyze each graph one by one:

Slave: mmap I/O vs file I/O

For all file sizes we see that mmap I/O is slightly slower than file I/O but not by a large margin. We suspect this is because allocating a new page when pages are filled is more costlya as the slave does not know the file size before the entire file is transmitted. For smaller file sizes the trend is not obvious since file transmission time has not yet dominate the total run time, which includes the creation of socket and other varibles.

Master: mmap I/O vs file I/O

For larger files mmap I/O is faster than file I/O since it does not have to copy the file into memory first, falling in line with our theoretical predictions. However, in terms of the total length of transmission time, the difference is relatively minor (especially under log scale where the larger values are compressed). For smaller file sizes the trend is not obvious as well due to the reasons previously mentioned. Also, for smaller files the one time cost of creating mmap may not be more efficient than the linear cost of copying to memory due to different constant overheads, much like sorting with HeapSort may not always be the fastest since we have to construct the heap. In this case, we see that at least in some occasions it is possible for file I/O to beat the transmission time of mmap I/O, which suggest that the constant overhead of mmap I/O is greater than file I/O.

File I/O: master vs slave and mmap I/O: master vs slave

The slave should generally be slower than master independent of the method we chose to transmit the file since it the master has to complete its transmission before the slave can completely receive the file. However, for smaller file sizes the transmission of file does not dominate the total runtime so the trend is not obvious. For larger file sizes, we see that the difference between master and slave is minor compared to the total execution time.

Contribution

Contribution is uniform among all group members.