# Lab 4 – Shared Memory Matrix Multiply

Draven Schilling

10-8-19

CS 3841

## Introduction:

In this lab we refined the processing speed of the previous matrix multiply lab in instead of using pipes to transmit data, to instead use shared memory blocks. This lab focuses on using mmap and shm_link to create shared memory blocks and demonstrates that using shared memory in our application is significantly faster than piping the data back from the child processes.

## Design:

For my design, I decided to build upon lab 3. I started by copying the createMatrix, deleteMatrix, and multiplyMatrix methods, then instead of using malloc I used mmap, and instead of free I used mumap. Form there, I was able to remove the piping infrastructure I previously used as well as the result collection loop (since the results are stored immediately in the child process). Overall, I think I had added 2 lines of code and just deleted a lot more; this made the process quite a bit smaller and more understandable.

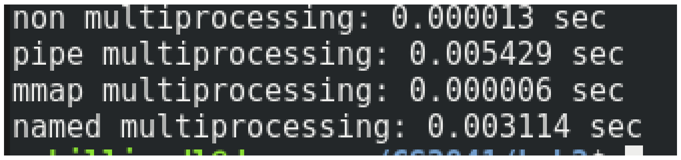
After the anonymous method was complete and tested, I moved to the named segment and my process was much the same. I copied the anonymous sections and added a snippet of code for shm_open that also took a name parameter in order to initialize a named memory segment. I also modified the deleteMatrix method to unlink the matrix with the name after mumap'ing the data section.

## Build Instructions:

1. Execute make to build the makefile
2. Run $./test matx maty (where matx is a standard matrix file and maty is another standard matrix file.
3. The results of the program will print matrix x followed by matrix y followed by the result matrix. It will also print out the associated time to calculate the result matrix using each of the approaches. Multiprocessing, non-multiprocessing, anonymous memory multiprocessing, and named memory multiprocessing.

## Analysis:

My results provided for matD * matD is:



```
non multiprocessing: 0.000013 sec
pipe multiprocessing: 0.005429 sec
mmap multiprocessing: 0.000006 sec
named multiprocessing: 0.003114 sec
```

Based on the speed it seems obvious the anonymous memory segment was by far the fastest. The named segment was faster the pipe() but still slower then the non-multi-processing and anonymous multiprocessing.

- Based on the amount of heap memory in use this makes sense. For both the pipe and named multiprocessing there were significant portions of allocated memory that was copied and freed in each subsequent child process. This was a lot more then the anonymous memory segment which needed to do virtually no copying. This used a lot less heap space and was one of the major factors in why it outperformed everything else.
- In terms of ease of use, the non-multiprocessing approach is the most straight forward because it emulates traditional programming practices. The anonymous multiprocessing was probably the easiest multiprocessing approach because it required the fewest lines of code and seemed more logical for how we would typically think of interfacing with multiple processes. The named segment was not that much harder than the anonymous segment and the pipe was somewhat harder because it dealt with the virtual file and reading from the child processes unintuitively.
- I would recommend using a shared memory segment when asynchronous operation is fine. That is, there is no competition and/or interference between child processes over parts of the shared memory. In our application this was the case and it made the most sense. But in applications where all the children might want to write to the same segment of shared memory, then using a pipe would be more logical for filtering and producing the desired result.

## Conclusion:

The most challenging part of this lab was the research. Because shm_open was not talked about in lecture, we had to do our own research on how to use it. I don’t think it was overbearing and/or too challenging, it just was the part that took me the longest. Overall, I think this lab was quite a bit easier then the last.

I cant think of any great direct feedback on what to improve, but I feel like I still don’t understand exactly the difference (in the background) between anonymous and named shared memory segments. I think additional lecture material on this would be useful.