Fall 2023 - CS 519 - Homework 1 (100 points, Due October 15th, 11:55pm)

This homework will consist of two parts: (1) using shared-memory for IPC, and (2) test your knowledge of caching. This is an individual homework.

CAUTION: Copying code from online sources or automatic generation tools like ChatGPT, Bard, and others may result in strict penalties.

Part 1 - IPC Shared Memory (50 points)

In this part, you will implement a pipe-based and shared memory-based IPC communication between two processes. The "HW-1" folder contains (1) IPC-pipe.c and (2) IPC-shmem.c.

Part 1.a IPC Message Passing Using Pipes (20 points) First, in IPC-pipe.c, you will add code that forks a child process (using fork()). The parent and child processes must communicate using Linux pipes and share work to perform matrix multiplication. The size of the matrix must be configurable using an input argument. Your code must allow multiple child processes to communicate and share work with the parent and other child processes.

Recall that for IPC, pipes use pipe(pipefd) to create a pipe, and use read(pipefd, buf, size) and write(pipefd, buf, size) for communication.

A tricky part is coordinating between the parent and child processes. Your code must synchronize them, albeit less frequently.

To synchronize between parent and child processes, you need some form of synchronization mechanism. A basic (i.e., not the best-performing) mechanism to use is a semaphore. For this part, you have the choice to use a semaphore or another locking mechanism.

To use semaphore (semctl and semop) system calls, we have added some dummy functions to initialize, reserve, and release semaphores. Here's a reference.

The matrix sizes should be large enough. We will test sizes up to 10000 x 10000 in CloudLab. So, make sure to allocate and release memory dynamically.

Your code must also check the correctness of the matrix multiplication output and print the time taken to perform the multiplication.

Part 1.b IPC Using Shared Memory (15 points) Next, in IPC-shmem.c, you will add code to perform matrix multiplication when parent and child processes communicate using shared memory by using calls like shmget() and shmat().

Your code must allow multiple child processes to communicate and share work with the parent process.

Part (1.c) Scalability Championship: IPC Scalability Points (15 points) A good solution must scale with the increase in the number of CPU cores. Remember the class discussions about CPU scaling, the related issues, and ways to address them. As one increases the number of CPUs, the cost of matrix multiplication should ideally reduce. Of course, there is a limit to CPU scaling.

As discussed in class, we will maintain a scoreboard, and the top 2 entries will get 15 points. The scores for other submissions will be based on how performant your code is related to the top-performing code. We will maintain a scoreboard on the class webpage.

You must clearly show a graph that shows the core count (on the x-axis) and the performance on the y-axis (similar to the Linux scalability paper). If your solution does not scale beyond a certain core count, you must explain why.

To improve scalability, you can use existing spin lock implementations. The repository below provides various spin lock implementations. Some of the spin lock implementations offer better performance. Try to understand which spin lock implementations provide better performance and why, and then make use of them! This is just a reference. You are welcome to consider other spin lock implementations.

`https://github.com/cyfdecyf/spinlock/tree/master`

Reporting Part 1 Results You should report in the following format and should match exactly as shown below. Please remove all your debug messages:

```
Input size: A columns, B rows
Total cores: N cores
Total runtime: X seconds
```

Part 2 - Cache Size Test (50 points)

Write a C/C++ program to determine the (a) L1, L2, LLC (last-line) cache size, (b) cacheline size (c) TLB size, and (c) memory latency (approximate) of a computer system.

You can use the starter file cache-tlb.c to get started. Note that you are not allowed to use existing Linux scripts or tools to extract this information.

You can use the function (currently empty) to add your code. Feel free to extend or change the arguments of the function.

- i. Report the results on CloudLab m510.
- ii. Clearly explain the limitations of your current code.

A submission that reports the correct numbers without explaining the insights used in obtaining them will not get significant credit. While writing the code you must be careful about microarchitectural enhancements such as out-of-order execution and prefetching that may affect memory system behavior.

Submission Instructions

- All executables must be compiled using a single Makefile. If you do not know how to write a Makefile, refer here
- Your submission must be compressed and submitted using Sakaai. tar -zcvf HW-1-.tar.gz HW-1

Computing Resource Please use CloudLab m510 for this assignment.

Starting Early This an essential homework for understanding the basics. Please start working on this homework early. If you have questions, make sure to ask them during office hours.