Time and Analysis:

When timing the executions, I kept track of the results of the ‘time’ for each process, searching for the same keyword in the same file. Here is a brief sample of my results:

|  |  |  |  |
| --- | --- | --- | --- |
| **Keyword** | **Unix Domain Sockets** | **Pipes** | **Shared Memory** |
| ‘beyond’ | real: 0.051s  user: 0.004s  sys: 0.008s | real: 0.027s  user: 0.004s  sys: 0.000s | real: 0.385s  user: 0.568s  sys: 0.060s |
| ‘John’ | real: 0.050s  user: 0.004s  sys: 0.004s | real: 0.025s  user: 0.004.s  sys: 0.000s | real: 0.396s  user: 0.568s  sys: 0.080s |
| ‘and’ | real: 0.157s  user: 0.004s  sys: 0.024s | real: 0.119s  user: 0.008s  sys: 0.004s | real: 0.407s  user: 0.568s  sys: 0.056s |

The first thing I want to discuss is the vastly larger user times found in the Shared Memory method. These larger times are due to the fact that the user time is summed over all cores being used. That is, since the Shared Memory program requires the use of multiple threads, and since the run time of each thread contributes to user time, we see what appears to be shockingly large user times. Another important thing to note is the slight disparity between the Unix Domain Sockets (UDS) and Pipes. The UDS are slower for two reasons. First, there is the necessity for there to be a connection to a socket with the UDS. In the implementation of the solution, there is essentially a busy waiting period while trying to get that connection to the socket, that is, the program continuously tries to build a connection to the socket until there is one and this takes up more time on the CPU (in the kernel space). In addition, using sockets is similar to network I/O which takes more time to send a message through which helps to explain the larger real time. So, since pipes can easily create file descriptors for reading and writing and the parent and child processes are able to read and write to the pipe that was created in their shared memory space before the forking, the Pipe program is able to complete the tasks faster.

To further explain some of the higher times that we see with the Shared Memory solution, the ‘saved time’ or the efficient portion of the solution comes in the actual processing of the lines in the text files. So, this program must pass the entire content of the file to its child process before the child process can start processing the lines from the file in parallel (concurrent processing of all of the lines). In contrast, both the Pipe solution and the UDS solution are able to have the parent process passing lines of the input file to the child while the child concurrently processes the lines being passed. In this way, these two solutions ‘save time’ or are efficient in their concurrent handling of reading the lines and processing the lines. This difference in concurrent reading and processing of lines versus concurrent processing of the entire file at once is what leads the shared memory solution to take a longer time. This may seem contrary to theory that suggests that shared memory is usually the most efficient IPC since its data passing doesn’t have to go through the kernel like what occurs with Pipes and UDS. However, as depicted, the solutions speed up the linear solution in different ways and thus we see the Shared Memory taking a longer time in this case.

IPC Methods in Linux:

Signals are the most basic form of IPC in Linux. Their primary use is to notify processes of changes in states or events that occur within the kernel or other processes. They can be used for communication and synchronization, but there are better alternatives for both.

Pipes provide a mechanism for one process to stream data to another. One end of the pipe is the read-end which is associated with a file-descriptor that can only be read, and the other end is the write-end which is associated with a file descriptor that can only be written. Pipes can be setup and used only between processes that have a parent-child relationship, and the communication is only one-way. Pipes are generally used to implement Producer-Consumer design amongst processes.

Named pipes (or FIFO) are a variant of pipes that allow communication between processes that are not related to each other (no parent-child relationship required). Additionally, the FIFO is actually a file on the disk that multiple processes can open, read, and write to. So, you would want to use a FIFO when you want to perform something similar to pipe operations of a file.

Message Queues are set up so that one process writes a message packet to the message queue and exits, then another process can access the message packet from the same message queue at a later point. The advantage of message queues over pipes are that the sender processes do not have to wait for the receiver processes to connect.

Shared memory allows one process to share a region of memory with at least one other process. This allows two or more processes to share data more efficiently with minimal kernel intervention. Good when working with large sets of data that need to be shared between processes.

Semaphores are locking mechanisms most commonly used to provide synchronization when processes are sharing resources. Ideal when dealing with multiple processes running through a critical section.

Unix Domain Sockets are sockets that can be used between processes on the same Unix system. Sockets themselves are like a pipe with two-way communications rather than one-way. Usually when Unix sockets are used, they're used with server and client programs, similar to Internet sockets.

MapReduce:

MapReduce is a programming model that allows for processing large sets of data via parallel computing. MapReduce consists of two functions that achieve this goal. The first function is "map", which takes the set of tuples and converts it into a series of intermediate key/value pairs. The second function is "reduce", which takes the output from the map function and merges the pairs in order to produce a smaller set of key/value pairs. MapReduce allows for high scalability and makes it easier to perform operations on large sets of data. It essentially splits the data and performs some operation on the data to create intermediate pairs that can then be grouped by the reduce function.

Hadoop:

Hadoop is an open-source MapReduce framework. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs. It gives users the ability to store and process huge amounts of data quickly. It also provides high scalability, high flexibility, and has a low cost. Hadoop is widely used in today’s age because there’s a massive amount of data out there to work with, and the amount is only increasing. Hadoop provides an accessible way for developers to manage large sets of data at a relatively low cost.