Special Issue Published in International Journal of Trend in Research and Development (IJTRD), ISSN: 2394-9333, www.ijtrd.com

Comparing and Evaluating the Performance of Inter Process Communication Models in Linux Environment

Ms. S.Krishnaveni and Ms. D.Ruby,
[1,2]Assistant Professor (SS),
[1,2]Department of Computer Science and Applications(M.C.A), Periyar Maniammai University,Vallam, TamilNadu, India

Abstract: In the software scenario, UNIX plays a vital role in implementing portable software architecture. Many different applications are based on UNIX platform. The performance evaluation of communication protocol is required to compare the feasibility. In this paper, we discuss the performance evaluation of various Interprocess Communication (IPC) mechanisms such as pipe, messages queue, streaming and datagram socket. The different IPC mechanisms are analyzed by comparing various sizes of data with a program simulating the messages across the network. Results were obtained for various sizes of data with Linux 2.2.5-15, FreeBSD 4.1, and FreeBSD 4.2.

To evaluate and compare various IPC methods, the source code for IPC was written in UNIX. All mechanisms are examined and evaluated for performance attributes such as memory, transfer rate, buffer sizes, data transfer methods and code complexity. The result shows that the Inter process communication procedures written in Linux 2.2.5-15 exhibited the best performance. Comparison of different mechanism shows that the streaming socket performs well. Various source code written to simulate the mechanism in different platform and its performance evaluation facilitate the full understanding of IPC at the source code level.

I. INTRODUCTION

The Unix operating system plays a prominent role in multi user platform. The functions and principles of operating system allow the user to run several processes simultaneously and share multiple resources such as power of CPU, memory, and other resources. Any none-trivial system developed on Unix systems will split the task into several subtask/sub processes and runs the processes simultaneously. The method of creating multi threads, starts, stop, communicate with each processes and synchronize them are built in the UNIX.

II. WHAT IS A PROCESS ?

A process can be defined as an entity that runs a small executable program. Each process have its unique process Identification number, execution stack, memory pages or blocks, and file descriptors table.

A process is a task of executing a small piece of code. A program may have a several processes executing simultaneously at the same time. For example, there is normally one copy of the 'tcsh' shell on the system, but for each separate user connections, there may be many ‘tcsh’ processes running. In this multi processes environment, the communication between different processes is exists, it is called inter process communication. In an IPC environment, several processes will try to execute the same piece of code, or will try to utilize the same resources. This situation of accessing a same resource by multiple processes at the same time is called 'Re-entrancy'.

Re-entrancy

The status of re-entrancy is defined as that a single process calls itself and execute the same code repeatedly. The concept of re-entrancy is also defined as that many processes try to execute the same piece of code in parallel. For instance, in a recursive method, the process executes the code defined in a function and the control of execution transferred to the same function again, ie. it calls the same code of execution again. During the function calls, the processes save the status information in a local variable.

In a multi-process environment, each process has a separate data section. So variables used in a process do not produce conflict among them. The distinct image of global variable of same program is available for two processes. Let us assume process A that runs program P and process B that runs the same program P, have distinct copies of the global variable 'i' of that program.

The most important problem in interprocess communication is managing simultaneous updation. For instance, consider the code, which “opens a file and write data on to it”. In the interprocess communication, if two processes try to run the same updation code , there will be a conflict occurs that which code is to be executed and which data is to be reflected in the file. Such situation can be managed by the “locking” mechanism, so that at a time one process is allowed to open a file and write the data into the file.

III. PROCESS CREATION

The fork() System Call

The fork() system call is a fundamental method to create a new process. It is also a very unique system call, since it returns twice(!) to the caller.

fork()

The fork() system call is used to split the main processes into two processes known as parent and child. When a fork() call is executed, the memory pages used by the original process get duplicated, so both parent and child process have the same structure of process. The differences of parent and child image occur at the call returns. The return value is the process ID of child process, when the call returns in the parent process. The return value is ‘0’ when it returns inside the child process. The return value is -1 (represents call fails), when the process has memory insufficiency/ too much processes/no new process is created. In case the process was created successfully, both child process and parent process continue from the same place in the code where the fork() call was used.

IV. CHILD PROCESS TERMINATION

In the UNIX multiprocess scenario, the environment have parent and child processes. There will be a possible situation that either

parent process exits before child or child exits before parent. The following states are possible in the parent and child process communications.

- When a child process exits, the signal is to be sent to the parent indicating the child's deaths state. After acknowledging, the child process and its status is removed from the process table. The duration of time between child exits and its acknowledgment to the parent is called childs "zombie" state.
- When a parent process terminates, the child process associated with parent is known as orphan process. An orphan process will be considered as child of 'init' process. The child process is automatically inherited by the 'init' process has the process number 1. The purpose of; init' process is to ensure that no child process is in "zombie" state. The written 'init' process properly acknowledges the death of its child process to the parent.
- When the parent process is not properly coded, the child remains in the zombie state forever. Such processes can be noticed by running the 'ps' command (shows the process list), and seeing processes having the string "<defunct>" as their command name.

A. The wait() System Call

The wait() system call is invoked when the parent process is required to know the status of child process. The state change of child is considered to be: the child process is terminated, the process is stopped, and the process is resumed. It is one of the way to acknowledge the parent process about the death of the child process. During the wait() is invoked, the process is suspended until one of its child processes exits. The call returns to exit status of child.

V. COMMUNICATIONS VIA PIPES

One or more related task may have the communication through pipe. In this manner, one task is dependent on previous task. So the new task is started from the earlier one as they are supposed to accomplish some related tasks.

A. What Is a Pipe?

A pipe is a special command in UNIX, used to control from where the input of command comes and where the output must go. It is used to connect two or more command together in a stream and control the input and output of the command. This mechanism considers the two processes such as ancestor and successor and sends a byte stream from one to other. The protocol must be carefully designed to utilize the pipe mechanism. The two way communication requires a parallel pipes to communicate.

The pipe protocol assures that the order, in which data is written to the pipe, is the same order as that in which data is read from the pipe. It assures that the data flow will be in the order from source to destination and no interruption occurs until one of the process exits.

B. The pipe() System Call

The pipe() system call has two types of file descriptors as an argument. The file descriptor refers to a pipe inode, and places them in the array pointed to by fields. filedes[0] is for reading, filedes[1] is for writing. The return value is 0 for the success condition, -1 is returned for error. The errno represents the status of error such as field is not valid, too many file descriptors etc.

C. Two-Way Communications with Pipes

In the multiple communication process, the two way protocol is used to communicate both directions, starts in parent to child and child to parent. The communication system what we require here is to open two pipes – one pipe from source to destination and other from destination to source. In this system of communication, the situation of arising deadlock is an unavoidable one.

D. Deadlock

During the inter processes communication, more than one processes are waiting for resources at the same time. The requested resources or event might be used by other processes in the same environment. The deadlock situation occurs when two processes communicate via two pipes. Here are two scenarios that could lead to such a deadlock:

- When two pipes are connected for two processes, the conditions such as both pipes are empty, but both processes are in a state to read data from their input pipes. Here each pipe will block on each other and thus in stuck situation.
- The second condition is more complex. Here two processes (A & B) are communicated via pipes. Each pipe has a temporary storage with a limited size of buffer. When a process A wants to write data on pipe A, it fills the data on the buffer A by write () call. The buffer is kept to read by the read process. When the buffer is full, the read operation is allowed to execute and write() system call will be blocked until the buffer gets free space. Both processes write operations will get blocked if the buffer is full and no read() occurs. Current two processes will be in deadlock.

VI. SYSTEM V IPC

The complex situation of inter process communication can be handled by invoking the mechanism called message queues, shared memory, and semaphores. The message queues mechanism is used to send and receive the messages among the inter processes, whereas the shared memory concept is used to allow the processes to share data in memory. The semaphore system is used to synchronize the process of resource access in multi process situation. It is a control variable that is used to control the access of common resources in a parallel communication system. The semaphore variable may change its status according to the condition specified by the programmer. The variable is used as a control variable to access the system resources.

VII. PERMISSION ISSUES

A. Private Vs. Public

The multiple processes can also be controlled or monitored by assigning privileges on access of resources. The access specifier is either private or public. Private access specification for a resource allows its own process or its child process to access whereas the public specification on a resource allows any process to access.

B. System Utilities To Administer System-V IPC Resources

As the inter process communication system is live outside the scope of a single process, the mechanism of maintaining the process status is required. The process of accounting deleted resources, crashed resources, number of exiting resources is needed to establish. Two utilities are used to administer the overall processes like 'ipcs' - to check usage of SysV IPC resources, and 'ipcrm' - to remove such resources.

The command 'ipcs' shows the utilization report for the resources. It gives the statistics such as identifier, owner, size of resources, and access permissions for various resources such as shared memory segments, semaphore arrays and message queues. Different unique report will be generated for each resource types. The command has flag representation to exhibit the particular type of resources. It can also be enterd by the user at the command 'ipcs'. The command which has '-m' refers the shared Memory segments, '-q' represents message Queues and '-s' for Semaphore arrays. The command like 'ipcs' with the '-l' flag is used to view the limits or size of the system and '-u' flag represents the memory usage statistics.

VIII. MESSAGE QUEUES

The way of establishing protocol is one of the problems with pipes. This protocol is based on sending separate messages. The pipe() system is based on byte stream. The input data stream from the pipe is needed to be converted in to packets before sending to the consecutive command. In the pipe processing system all the processes are executed in FIFO manner, that the processes are executed in the order they arrived. Priority or intermediate accessing is a difficult task. The intermediate process must wait until the entire proceeding task to complete its processes. This means that before reading any part of the stream must consume all the bytes sent before the piece you're looking for, and thus it is needed to construct queuing mechanism on which data can be placed.

A. Creating A Message Queue – msgget ()

The msgget() system call is used to initiate a message queue. The command has two attributes of parameters such as a queue key, and flags. The key is one among the following:

- IPC_PRIVATE - used to create a private message queue.
- A positive integer - used to create (or access) a publicly-accessible message queue.

The second parameter contains flags which are used to denote the control on which the system is processed.

IX. BACKGROUND - VIRTUAL MEMORY MANAGEMENT UNDER UNIX

To achieve virtual memory, the system divides memory into small pages each of the same size. For each process, a table mapping virtual memory pages into physical memory pages is kept. When the process is scheduled for running, its memory table is loaded by the operating system, and each memory access causes a mapping (by the CPU) to a physical memory page. If the virtual memory page is not found in memory, it is looked up in swap space, and loaded from there (this operation is also called 'page in').

When the process is started, it is being allocated a memory segment to hold the runtime stack, a memory segment to hold the programs code (the code segment), and a memory area for data (the data segment). Each such segment might be composed of many memory pages. When ever the process needs to allocate more memory, new pages are being allocated for it, to enlarge its data segment.

When a process is being forked off from another process, the memory page table of the parent process is being copied to the child process, but not the pages themselves. If the child process will try to update any of these pages, then this page specifically will be copied, and then only the copy of the child process will be modified. This behavior is very effcient for processes that call fork () and immediately use the exec () system call to replace the program it runs.

X. COMPARISON MODEL

An overview of actual transaction processing system built by using System V IPC. The name space used by System V IPC is an advantage not a problem if we use file descriptors. Because identifiers allow the process to send the message to a message queue with a single function call. Bulk of data transferred from one process to another process. We have created a bench mark program using pipes and message queues. In both the cases, the various sizes of data are inputted for both the programs and we have analysied the performance and response rate. The size of the data various from bytes to Maga Bytes.

XI. RESULT ANALYSIS

The bi-directional flow of data between the process using the message queues and pipes are analyzed by giving various sizes of data. The test consisted of the program that created IPC channel called fork and sent various Bytes ,KiloBytes and Maga Bytes data from parent to the child. Data was sent using various calls to msgsend, with a message length of various byte sizes, for the message queue, and various calls to write, with the length of various bytes size for the stream pipe. We got the timing comparision table for pipes, message queues and sockets.

Table 1: Timing comparison of two models

		Pipes		D1
S.No	Data Size	Start time	End time	
1.	32 bytes	20:21:22	20:21:22	0
2	306 bytes	20:22:01	20:22:02	1
3.	3.0 KB	20:23:38	20:23:40	2
4.	30 KB	20:24:00	20:24:01	1
5.	270.4 KB	20:24:20	20:24:23	3
6.	1.6MB	20:24:46	20:24:50	4
7.	33MB	20:25:05	20:26:00	55
8.	99.0 MB	20:26:25	20:29:06	161

Table1. Timing analysis of Pipes

Datagram sockets			
Data Size	Start time	End time	D4
32 bytes	21:03:12	21:03:13	1
306 bytes	21:07:31	21:07:32	1
3.0 KB	21:09:07	21:09:08	1
30 KB	21:04:55	21:04:56	1
270.4 KB	21:05:22	21:05:23	1
1.6MB	21:05:51	21:05:52	1
33MB	21.06:15	21:06:16	1
99.0 MB	21:06:46	21:06:48	2

Table2. Timing analysis Of Pipes

Streaming sockets				
S.No	Data Size	Start time	End time	D3
1.	32 bytes	20:49:23	20:49:24	1
2	306 bytes	20:50:21	20:50:22	1
3.	3.0 KB	20:58:07	20:59:27	1
4.	30 KB	20:00:38	20:00:39	1
5.	270.4 KB	21:01:55	21:01:57	2
6.	1.6MB	20:56:34	20:56:36	2
7.	33MB	20:57:08	20:57:11	3
8.	99.0 MB	20:58:15	20:58:21	6

Table 3: Timing analysis of Streaming Sockets

Streaming sockets				
S.No	Data Size	Start time	End time	D3
1.	32 bytes	20:49:23	20:49:24	1
2	306 bytes	20:50:21	20:50:22	1
3.	3.0 KB	20:58:07	20:59:27	1
4.	30 KB	20:00:38	20:00:39	1
5.	270.4 KB	21:01:55	21:01:57	2
6.	1.6MB	20:56:34	20:56:36	2
7.	33MB	20:57:08	20:57:11	3
8.	99.0 MB	20:58:15	20:58:21	6

The user, System and clock for the 20 Mega bytes of data is analysis with pipes and message queues. The times are all in seconds.

Table2. Comparison of user ,system, Clock Times

IPC	SVR4		
	USER	System	Clock
Message Queues	0.7	19.6	20.1
Pipes	0.5	21.4	21.9

Data was sent using various calls to write, with the length of various bytes size for the stream pipe. We got the timing comparison table for pipes. The following graph figure 1 represents the performance of pipes for various sizes of data mentioned in table1. The performance curve states that response time varies when there is a bulk of data inputted.

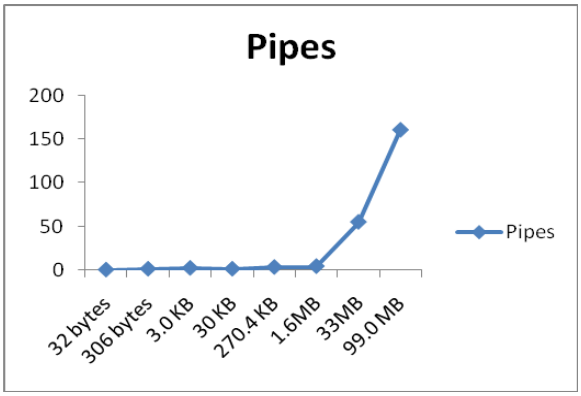

Figure 1. Response time for pipes

Data was sent using various calls to msgsend, with a message length of various byte sizes, for the message queue. The following graph figure 2 represents the performance of message queues for various sizes of data mentioned in table1. The performance curve states that response time varies when there is a bulk of data inputted.

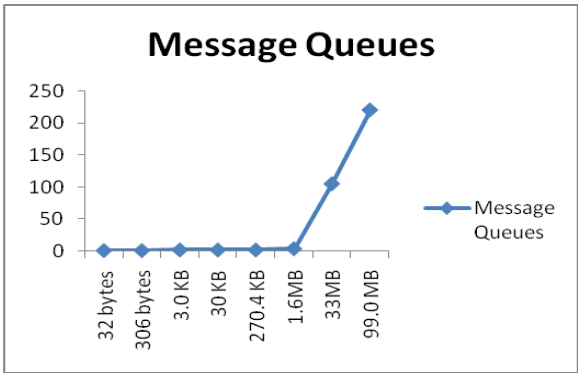

Figure 2. Response time for message queues

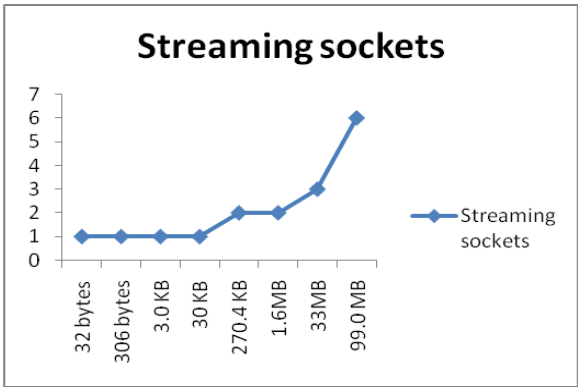

Figure 3. Response time for Streaming sockets

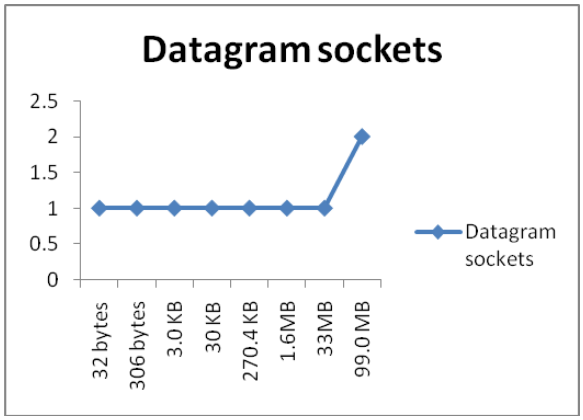

Figure 4. Response time for Datagram Sockets

Figure5. states that ,when we compare the response time of pipes and message queues, for smaller size of data message queues perform better than pipes. When the data size increases, the performance of the pipes is better than message queues. When the data size is 99.0 MB the response time for pipe is 161 seconds and for message queue is 220 seconds and streaming sockets is 6 seconds and datagram socket is 2 secongs

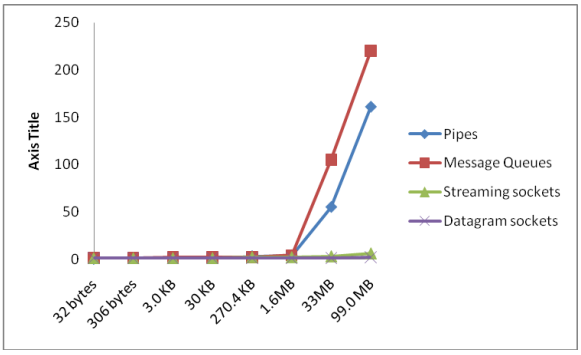

Figure 5: Comparison of response time for pipe, queue and sockets

CONCLUSION

The fundamental problem System V IPC is that the IPC structures are system wide and donot have reference count. For example if we create a message queue, place some message on the queue and them terminate, the message queue is not deleted. They remain in the system until specifically read or deleted: by some executing ipcrm(1) command or by the system being rebooted . Compare this with the pipe which is completely removed when the process to reference it terminates. The message queues are that they are reliable, flow controlled, record oriented and can be processed in other than first in first out order. The streams also possess all these properties, although an open is required before sending to a stream and close required when we are finished. Both message stream and pipes are connection less. The message types are identified by the priorities. We compare the reponse time of pipes, message queues,streaming sockets and datagram sockets for small and large size of data datagram sockets perform better than streaming sockets. But for the bulk size of data datagram sockets overcomes the message queues, streaming sockets and pipes.

References

[1]. Comparing Some IPC Methods on Unix by Gene Michael toverSunday, 27 January 2002

[2]. Mike Gancarz, The Unix Philosophy. (1995) Digital Press; Newton, MA. ISBN 1-55558-123-4.

[3]. Performance analysis of five interprocess communication mechanisms across UNIX operating systems Patricia K. Immich, Ravi S. Bhagavatula and Dr. Ravi PendseDepartment of Electrical and Computer Engineering, Wichita State University, 1845 Fairmount Box 44, Wichita, KS 67260, USA 3 May 2003

[4]. www.ebooknetworking.net/ebooks/unix-kernel-interproce ss-communication-parameters.html

[5]. Named Pipe Security, Interprocess Communitions: Platform SDK, MSDN Library, January 2001.

[6]. Berkeley UNIX System Calls and Interprocess Communication, January 1987.

[7]. R.Klefstad, UNIX Network Programming, Introduction to UNIX Local and Remote Interprocess Communication,