CSCanada
Management Science and Engineering
Vol. 5, No. 3, 2011, pp. 162-164
DOI:10.3968/j.mse.1913035X20110503.052

ISSN 1913-0341[Print]
ISSN 1913-035X[Online]
www.cscanada.net
www.cscanada.org

The Analysis and Comparison of Inter-Process Communication Performance Between Computer Nodes

ZHANG Xiurong[1,*]

[1]Institute of Media, Inner Mongolia University for Nationalities, Tongliao, 028043, China.
[*]Corresponding author.
Address: Institute of Media, Inner Mongolia University for Nationalities, Tongliao, 028043, China.
Email: zhangxiurong6413@163.com

Received 10 June, 2011; accepted 6 August, 2011

Abstract
According to the physical location of communication processes in web-based software system, there are usually two types of communication processes, namely the communication at different nodes and the communication between different processes. The former is mainly achieved by internet domain sockets, while the later is achieved by pipes, shared memory and sockets provided by most operating systems. We therefore studied theses three kinds of technologies and evaluated their communication performance.

Key words: Process; Communication; Machine communication; Comparison

ZHANG Xiurong (2011). The Analysis and Comparison of Inter-Process Communication Performance Between Computer Nodes. *Management Science and Engineering*, 5(3), 162-164. Available from: URL: http://www.cscanada.net/index.php/mse/article/view/j.mse.1913035X20110503.052
DOI: 10.3968/j.mse.1913035X20110503.052

INTRODUCTION
According to the physical location of communication processes in web-based software systems, there are usually two types of communication processes, namely the communication at different nodes and the communication between different processes. The former is mainly achieved by internet domain socket, while the later is achieved by pipes, shared memory and sockets provided by most operating systems. Many network applications such as intrusion detection systems (IDS) and network management system require very high level of real-time performance. Under such circumstances, the quality of communication performance will directly determine the applicability and efficiency of a system. Among these communications, the inter-process communication within a node[1] usually accounts for a large proportion. Therefore, to study its performance is very important.

The network programming usually involves the interaction between various processes and coordinates their activities through some active or passive inter-process communications[2].

Linux systems provide a variety of inter-process communication methods, mainly including signals, anonymous and named pipes, message queues, semaphores, shared memory and memory mapping, and UNIX domain sockets. In this paper, we focus on pipes, shared memory, and UNIX domain sockets and introduce the main method of each technology and the principle of how the inter-process communication is achieved. We then test the inter-process communication performance of each of the aforementioned method.

1. PIPES

1.1 The Definition of Pipe
In order to accomplish a task, data sharing is bound to happen during the cooperation of two or more processes. It is not suitable to transfer a large amount of information from one process to another, even if the signals are very useful for handling abnormal events or errors. To solve this problem, UNIX provides a technology called pipe. Pipe is often used as a one-way communication channel that connects one process with another[3]. The process on one end uses write calls to send data through the pipe while the process on the other end uses read calls to receive data.

1.2 The Type of Pipes

Pipe provides a simple and synchronous transmission of information between processes. It can be divided into two categories: anonymous pipe and named pipe. An anonymous pipe can only be used between related processes, i. e., between parent and child processes or between two sub-processes. In addition, it should exist together with the creating process. A named pipe has directory entries to access files. Therefore, it can be used between un-related processes.

1.3 The Implementation of Pipeline Technology

Pipes use first-in-first-out (FIFO) method to save a certain amount of data. When the pipe is opened, a process writes in one end and another process reads from the other end of pipe. The main process uses fork () to create a child process so that both parent and child processes can simultaneously read and write through the same pipe. Because the pipeline must be one-way, the direction of data flow must firstly be determined and then the unnecessary commands should be turned off. Thereafter, you can use the read () and write () to read and write. The steps of inter-process communication using anonymous pipes are as follows:

(1) Create necessary pipelines.

(2) Generate one or more child processes.

(3) Close / copy the file descriptors in order to link to the relevant end of pipe.

(4) Close unwanted ends of the pipes.

(5) Perform communication activities.

(6) Close all remaining open file descriptors.

(7) Wait for child process to finish.

When the read () or write () function is performed, it can block the pipeline operation, which ensures the write process prior to the read process. As a result, the parent and child processes become synchronized.

2. SHARED MEMORY

2.1 The Principles of Shared Memory

Shared memory is a mapping of virtual memory that is mapped and shared between various processes. With this technology, different processes can simultaneously access the same data stored in shared memory and read / write through a certain proportion of shared address space, thereby achieving a direct communication with each other. To some extent this is the fastest but not the easiest way of process communication. These data can be accessed randomly. To avoid the inconsistency of data, semaphores are often introduced to coordinate the accesses to shared memory segments.

Because multiple processes share a physical memory space without unnecessary data replication, it can improve the inter-process communication performance. However, the synchronization and ejection of replication must be addressed[4].

2.2 The Implementation of Shared Memory Technology

(1) Use a process to create / allocate a shared memory segment and set the size and access rights during the creation.

(2) Attach the process to this memory segment and map it to the current data space simultaneously. Each process can access the shared memory segment through its articulated address.

(3) When the process has finished the operation on the shared memory segment, it can be disconnected from the segment.

(4) Operate the shared memory segment by the process.

(5) When all processes complete the operation on the shared memory segment, they are usually deleted by the process that creates the segment.

3. SOCKETS

3.1 Socket Communication Theory

Socket is an abstractive data structure provided by the operating system to create a channel (connecting point) in order to send and receive messages between unrelated processes[5]. It actually provides an endpoint of inter-process communication, which enables the completion of data transfer.

Socket is designed for the client - server model. Different socket system calls are provided for the client and server programs. Linux sockets include stream sockets (sock-STREAM), datagram sockets (sock-DGRAM), and row sockets (RAW-SOCKET).

The basic model of programming using socket is the client - server model. The server calls socket () first to create a new socket, and then use bind () to bind the socket to the client port. Thereafter, the server calls listen () to set the length of listen queue in order to prepare for accepting the request from clients; and then it listens to the request from the clients at connecting ports with listen (). The processes in client end create certain types of socket through the socket () and call host servers for a connection request with connect (). While connected, the host's IP address, socket, and other relevant information must be provided. The inter-process communication can be achieved using the socket method.

3.2 The Implementation of Socket Technology

The working procedures of server processes:

(1) Create a socket.

(2) Configure IP address.

(3) Bind the IP address to the socket.

(4) Listen to the request of connection from other processes.

(5) Accept the connection request.

(6) Creates a child process to receive the data transmitted from the client process and locate it in the designated buffer.

(7) Appropriate data processing.

(8) Wait for all child processes to finish and then exit.

The working procedures of client processes:

(1) Create a socket.

(2) Obtain the server's IP address.

(3) Requests a connection to the server.

(4) Send data.

(5) Receive response signals.

(6) Exit.

4. THE EVALUATION OF COMMUNICATION PERFORMANCE BY THREE METHODS

4.1 Test Procedures

The hardware used for performance test included eight Intel Pentium 4 personal computers (CPU 2.0 G, Ram 1 G, operating system redhat Linux 9.0), connected via 100Mbps Ethernet. Test procedure was based on anonymous pipes, shared memory, and UNIX domain socket technologies. The program was designed with C language. The aim was to test the machine performance of inter-process communication at a user level.

The standard procedure was used to test the communication performance of anonymous pipes, shared memory, and socket method. In each test the system had the same resource usage with a CPU utilization of 35%. During the test each packet with different length was tested for 30 times and the average value was obtained as the final result. Transmission time was defined as the duration from the time when the information of main task was sent out to the time when the response signal was received.

4.2 Performance Analysis

According to the test results of communication performance, the transmission time of pipeline was basically unchanged regardless of the amount of data transferred. It was time-consuming to establish a pipeline. But once it was established, the data transmission time was basically the same regardless of the amount of data. However, the data transmission time was increased with the increase of the number of bytes transferred by shared memory and sockets. Therefore, pipeline was the best method when a large amount of data needed to be transmitted. When the amount of data transferred was less than 4600 bytes, shared memory had obvious advantages in transferring data with a very fast speed. However, when the data transmission amount exceeded 4600 bytes, such advantage was eliminated because it had to introduce semaphores increasingly to coordinate the accesses to a shared memory segment. As a result, its communication performance became poorer than other two methods. When the data transmission amount was 5100-12000 bytes, the communication performances of pipes and shared memory were not as good as UNIX domain socket.

REFERENCE

[1] Wu, G. (2009). *Linux Basic Study* (2nd edition). Beijing: Tsinghua University Press.

[2] Cohoon, J., Davidson, J., et al. (2005). *Java Programming*. Beijing: Tsinghua University Press.

[3] What is the Computer Pipeline Technology [EB / OL]. Retrieved from Http://www.01ruodian.com/school/news.asp?id=10009.

[4] Gao, S. C., Mao, D. L., & Cao, X. (2008). *Computer Network Tutorials*. Beijing: Higher Education Press.

[5] Lee, Y. S., Lu, J. W., & Cao, X. (2003). *Java Language Programming for a Layman*. Beijing: China Youth Publishing House.

