DIPC: A System Software Solution for Distributed Programming

Kamran Karimi
Department of Computer Engineering
Iran University of Science & Technology
Narmak, Tehran
Iran

E-mail: karimik@sun.iust.ac.ir

Mohsen Sharifi
Software Engineering Laboratory
Department of Computer Engineering
Iran University of Science & Technology
Narmak, Tehran
Iran

E-mail: mshar@rose.ipm.ac.ir
Phone: (0098-21) 745-6783

Abstract

Distributed Inter-Process Communication (DIPC) provides the programmers of the Linux operating system with distributed programming facilities, including Distributed Shared Memory (DSM). It works by making UNIX System V IPC mechanisms (shared memory, message queues and semaphores) network transparent, thus integrating neatly with the rest of the system. The underlying network protocol used is TCP/IP. DIPC is targeted to work on WANs (Wide Area Networks) and in heterogeneous environments.

Keywords

distributed programming, parallel programming, multi-computer, IPC, DSM, Linux, WAN, TCP/IP, heterogeneous environment

Introduction

Considering the limitations in increasing the processing speed of uni-processor systems, using multiple processing elements (PEs) to solve a problem is becoming the solution of choice among computer users [1]. Here solving a problem requires the following abilities:

- Starting programs on different PEs.
- Providing them with the needed data and collecting the processed results.
- Synchronizing them, so that they access the data and results in an orderly manner.

There are two main trends in designing *parallel systems*: Multi-Processors (MPs) and Multi-Computers (MCs) [1,8]. MPs use processors with access to some shared memory to do their work. This shared memory makes the programmer’s job in transferring data and synchronizing between the processors easy [2,3]. Usually the operating system undertakes the task of running parts of a parallel application on different processors. They usually need custom designing and are hard to reconfigure. MCs on the other hand, are mostly built by connecting some stand-alone computers over a network. These are also known as distributed systems.

There are differences between distributed and parallel software, though a distributed program may also be a parallel one.

Considering the fact that each of the computers in an MC can in itself be an MP, it can be concluded that the MC architecture is an improvement over, rather than a contender to the MP architecture. MCs are easy to design and reconfigure, but the fact that different computers don’t have direct access to each other’s memory, makes their programmer’s job much harder. This includes both data transfer and synchronization. Much work has been done on remote program execution, one example being the ‘rsh’ UNIX command. A system with the ease of use of MPs and the ease of the design and reconfiguration of MCs would be highly desirable.

One interesting method to ease an MC programmer’s burden is to simulate a shared memory accessible to different computers in the network. This is called Distributed Shared Memory (DSM) [1] and is usually implemented through software means. Here the programmer has the ease of use of an MP solution, while taking advantage of the benefits of an MC architecture. The price paid for this, most of the time, is a drop in performance. Careful programming can diminish this negative factor. In practice the advantages of MCs are so compelling to many people that they seem to be the trend of the future.

Motivation

UNIX is among the platforms of choice for writing parallel and distributed programs. The AT&T UNIX provides what is known as System V IPC mechanisms, consisting of shared memories, message queues and semaphore sets, to enable programmers to exchange data and synchronize between processes running on the same computer [4,5]. These could be used in an MP system. Different processes use the same numerical “key” to gain access to the same IPC structure. System V IPC is well documented, familiar to many programmers and widely available in different UNIX systems. Many programs use it to communicate between different processes. However, UNIX does not provide enough standard facilities for distributed software development. Programmers and users of different UNIX flavors have to resort different add-on packages and various programming models to write or use distributed software. They may be forced to use some specific programming languages (e.g. Linda) and methodologies (e.g. Object Orientation). The mechanisms provided by these packages usually differ greatly from one another, each requiring the users to learn some new material, which won’t be of any use to them when migrating to other methods. Many also require detailed involvement of the programmer or the user in the process of transferring information over the network. One example is the PVM (Parallel Virtual Machine) software [10]. The need for a simpler distributed programming model under UNIX, usable by more programmers, is clearly present.

Introducing DIPC

DIPC (Distributed Inter-Process Communication) is a software-only solution for creating a multi-computer using ordinary and readily available PC computers and networking hardware. Each node could be an i386 or higher machine. The software used is a modified version of the Linux operating system, with TCP/IP [7,8] as the networking protocol. The main goal is to enable programmers to write distributed software quite easily. It tries to ease data transfer and synchronization operations between processes in different computers by making UNIX System V IPC mechanisms network transparent.

DIPC is being developed in Iran University of Science & Technology's Department of Computer Engineering,

DIPC Goals

The following goals were in mind when designing and implementing DIPC:

- Simplicity of the system.
  Preferring simplicity of the algorithms whenever a conflict between that and the performance arises (This design decision may change in future versions).

- Transparency of the distributed facilities.
  Doing distributed actions should not be very different from doing the same actions in a single computer.

- Independence from network characteristics.
  The programmer should not be concerned with physical characteristics of the computer network, such as network topology, addresses, etc.

- Compatibility with legacy software.
  Non-distributed programs using System V IPC mechanisms should be able to co-exist with distributed programs.

- Simplicity of programming.
  Preserving the UNIX semantics would helps those programmers who are already familiar with UNIX, and prevents the need to master some completely new programming models. Others could benefit from the wealth of information already available about System V IPC

- Independence of any specific programming tool or model.
  Programmers should be able to use DIPC in any language that can access operating system's functions. (S)He is not limited to any specific language or software engineering methodology.

- Ability to turn legacy programs into distributed ones.
  It should be relatively easy to change older programs, using System V IPC mechanisms and probably running on multi-processors, to take advantage of DIPC, thus making them distributed programs.

- Ability of the programmer to influence program performance.
  The main performance parameters (frequency and amount of data exchange between machines) should be in the hands of the programmer.

- Ability to develop programs on inexpensive hardware.
  Programs could be developed on a single computer and later used in a computer cluster.

- Running DIPC system code in user space as much as possible.
  This is the trend followed in Microkernels, though Linux is a monolithic operating system.

- Making DIPC work on WANs (Wide Area Networks).

- Making DIPC work in a heterogeneous environment.

Discussion

With DIPC, programs on different computers can use the same IPC key to gain access to the same IPC structure. This frees the programmers of learning new methods and integrates quite naturally with the rest of the system. Making DIPC services available at the operating system level allows other programs to use them without regard to the language or the methodology used during the software construction, and there is no need to learn completely new ways to do distributed computing.

Distributing System V shared memory provides Linux with DSM capabilities. Processes on different machines can read the shared memory at the same time, and the effects of a write will automatically become visible to others. Under DIPC, strict consistency [6] is used as the consistency model of the shared memory. This means that the readers of the shared memory get the most recently written values. This is very familiar to programmers. Programs can use the shared memory as an asynchronous way of exchanging data, or they can use System V messages to transfer data synchronously. Semaphores are used as the arbitration mechanism for access to the shared memory or message queues.

It should be noted that DIPC provides a set of mechanisms, and is not concerned with policies. The software designer determines how these mechanisms are used.

DIPC is targeted to work in heterogeneous environments, consisting of a network of machines with various architectures (e.g. Motorola 68K and Intel x86 computers). It supports the programmer in the task of converting information between different data representation formats. Using a WAN is also an important feature. Programmers may use machines on the Internet to accomplish a given job.

DIPC creates the notion of *configurable logical clusters* in a physical network, enabling different groups of computers to collaborate on a job without disturbing other machines on the network, which may be executing the same programs.

Current status of DIPC

DIPC is currently in the alpha stages of development. A port to Linux for Motorola 680x0 processors is underway. DIPC sources and the related documents can be found on the Internet via anonymous FTP at sunsite.unc.edu in /pub/Linux/Incoming, or /pub/Linux/system/Network/distrib/dipc. Alternatively, they can be directly obtained from the authors.

Summary

A Simple distributed programming system was briefly introduced. The services provided by this system are extensions to previously existing mechanisms. This is done in order to ease the application programmer's job. It was noted that by integrating these services within the operating system, other layers of the system can use them transparently.

References

[1] Andrew S. Tanenbaum, *Distributed Operating Systems*, Prentice-Hall, 1995.

[2] Bruce P. Lester, *The Art of Parallel Programming*, Prentice-Hall, 1993.

[3] Robert G. Bab II, *Programming Parallel Processors*, Addision-Wesley, 1988.

[4] Maurice J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall, 1986.

[5] W. Richard Stevens, *Advanced Programming in the UNIX Environment*, Addison-Wesley, 1992.

[6] Bill Nitzberg and Virginia Lo, *"Distributed Shared Memory: A survey of Issues and Algorithms"*, Computer, August 1991, pp. 52-60.

[7] Andrew S. Tanenbaum, *Computer Networks*, Prentice-Hall, 1989.

[8] W. Richard Stevens, *UNIX Network Programming*, Prentice-Hall, 1990.

[9] Kai Hwang, *Advanced Computer Architecture*, McGraw-Hill, 1993.

[10] PVM web page: http://www.epm.ornl.gov/pvm/pvm_home.html