International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-2S10, September 2019

Implementation of Improved Synchronization in Inter Process Communication using Threads for Microkernel and Distributed Operating Systems

Sukhvinder Singh Bamber

Abstract*: *Interprocess Communication (IPC) is used by the cooperating processes for communication and synchronization. With the advent of Distributed Systems and Microkernel Operating systems, IPC has been used for designing the system for cooperation. This raised the requirements for improving the communication and synchronization for the better performance of the system. Here, a mechanism of synchronization between the processes to reduce the waiting time of process using POSIX (Portable Operating System Interface) threads has been proposed to perform and synchronize the given task.

Keywords* : *IPC, Microkernel Operating System, Distributed Operating System, POSIX.

I. INTRODUCTION

Interprocess Communication is the mechanism that makes the processes to communicate and synchronize actions for communication. For example, a web browser demands a web page from a web server, HTML data is returned back to the client web browser by a web server serving the request. Basic IPC structure includes two operations: send(message) and receive(message). Cooperating process can send the message which can either be of fixed or variable size. Also the communication link must exist between the cooperating processes if the cooperating processes want to communicate. These communication links can be unidirectional or bidirectional. The transfer of data among the processes usually makes use of sockets like telephone connection [1]. There are two types of processes: Independent and Cooperating. A process which cannot affect parallel processes or get affected by the other executing processes or the process which cannot share data with other processes in execution is called the independent process. A process is termed as cooperating if it can be affected or can affect other processes executing parallel in the system [2]. The message passing system is widely used for communicating data for the distributed systems. For the microkernel Operating Systems this model can be implemented by using shared memory or pipes.

To implement this model on a distributed system, we need to use the sockets for communication. For the microkernel operating system, any of the five methods: FIFOs, Mapped Memory, Pipes, Shared Memory and Sockets for IPC can be used [1]. Shared Memory is a mechanism in which multiple executing processes in a system can access the same block of memory which turns out to be a shared buffer for the cooperating processes to communicate parallel with each other. Memory Mapped method is a mechanism in which a file is mapped onto RAM and can then be modified by directly changing memory address instead of sending it to a stream. This mechanism is equally beneficial as a standard file. Pipe is another mechanism in which data is written at the writable end of the pipe and then the operating system buffers it till it is read at the readable end of the pipe. Full duplex data streams between the cooperating processes can be implemented by creating two pipes utilizing standard input and output. Socket is a mechanism of transferring the data through the network interface to a parallely cooperating process on the local computer or to another computer in the network.

Every type of function is implemented using threads. A thread is the smallest set of instructions (programmed) that can be executed independently by a scheduler in execution, which further is the part of an operating system [3]. Execution/implementation of processes and threads vary in one operating system to another, but in most of the cases a thread is a part/component of a process in execution [4].

POSIX thread, also referred to as pthread, is an executing sequence of instructions that executes independently from the platform/language in which developed as well as a parallel execution model. It enables the executing program to control parallel multiple different execution flows of work which may overlap the time allocated. Each of this workflow is called a thread. The execution and control of these workflows is achieved by calling the POSIX Thread API (Application Programming Interface). POSIX thread is an API defined in the standard POSIX 1c. thread extension (IEEE Std 10031c-1995) [6]. Here, a mechanism of synchronization between the processes to reduce the waiting time of process using POSIX threads has been proposed to perform and synchronize the given task.

Revised Manuscript Received on September 25, 2019

* Correspondence Author
Sukhvinder Singh Bamber, Assistant Professor**,** Computer Science & Engineering, University Institute of Engineering (UIET), Panjab University SSG Regional Centre, Hoshiarpur, Punjab, India.

II. IMPLEMENTATION

The synchronization model can be implemented using threads for two types of systems: Distributed Systems and Microkernel Operating Systems. In the Distributed Systems there may be many servers but for the simple explanations of the model one client system and two server systems are taken. And, in the Microkernel Operating System, there will be one user interactive application and two systems are taken. The explanations for implementation for both of the systems are given below separately:

III. DISTRIBUTED SYSTEMS

For the sake of simplicity and better understandability: one client system and two server systems are considered. Every type of system has its own operating system. An operating system always either does something or waits for an input from user. Let's consider that the client system has an application which directly interacts with the user. As soon as user gives the command two threads will be created: one thread takes the data as input (data on which the operation is to be performed) from the user and another initializes the server application which has the definitions of the procedures to perform the task commanded by the user. Let's consider this as a Server: S1. If the server application installed at S1 can perform the complete task then it is performed and output is returned back to the requesting Client System and the application S1 is terminated. Else if S1 has the application program which can partially perform the task and some procedures required to complete the remaining task is in the server application program installed at Server2: S2, then again two threads are created on the S1: one thread will perform the operation partially and generate a new data and another initializes the server application installed at S2 which has the procedures to complete the task.

Eventually, on the S2 task will be finished and output is returned to the S1 and then S2 application is terminated. Finally, S1 returns the output to the client system and S1 application is terminated.

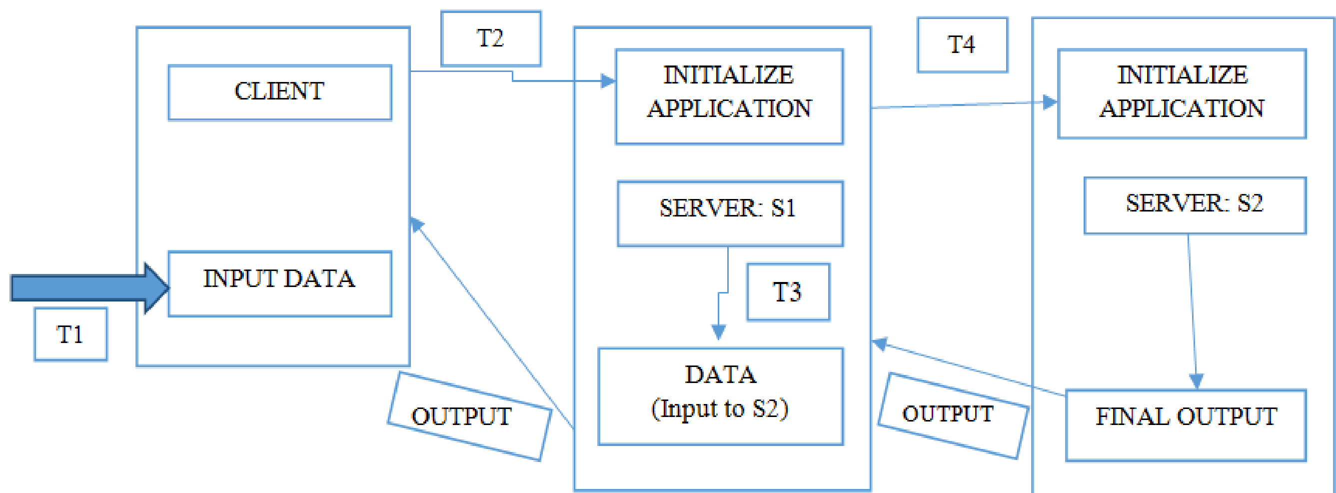

Figure 2.1: Implementation of working model for distributed systems (T1-T4 are pthreads).

IV. MICROKERNEL OPERATING SYSTEM

In a monolithic kernel (GNU/Linux) the complete kernel runs under a single process. For every particular task a child process is created using clone() system call. In a monolithic kernel, modularity is achieved by the dynamic loading of modules called hot-plugging. But the monolithic kernels cannot be used in the real time operating systems. In this type of operating systems, microkernels are used. So there are many processes for a single kernel. Here for improving the performance of operating system, communications between the processes (IPC) has to be improved. This paper presents an improved model of synchronization between the processes. Let there be a process that directly interacts with the user (usually command line interface or graphics user interface) and two other processes having the procedures to perform the tasks. The user interactive application is always in a waiting state to take the input from the user. When a command is given by the user two threads are created: one takes the input data from the user, another initializes the process that has the procedures to perform the task. Let's consider it is Process: P1. Now if task can be performed only by the P1 then it is performed and output is returned to the user interactive process and P1 is terminated. Else if P1 does not have enough procedures to perform the complete task then in P1, two threads are created. First thread initializes another process (P2), performs the partial task and generates a new data as an input to the P2. The P2 performs the remaining task to generate the final output. The output is returned to the P1 and then the process P2 is terminated. Now P1 returns the output to the user interactive application and is terminated.

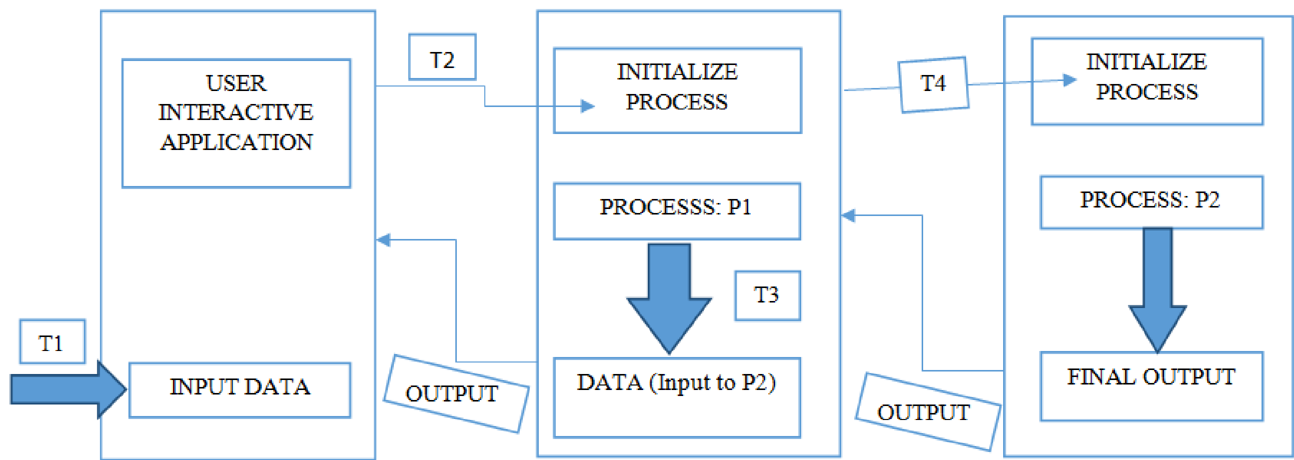

Figure 2.2: Implementation of working model for microkernel operating systems (T1- T4 are pthreads)

V. CONCLUSION

By using threads we can initialize the application the data is available. One thread initializes the process to perform the task and another thread makes a container to receive the data. As soon as the data is available, the application program (contains the function implemented as thread function) is initialized. So the application program does not need to wait.

This model can be used to design a client operating system which will make the Remote Procedure Calls (RPCs) and server operating system for the distributed systems. The procedures are stored at the servers. If these procedures are dependent on other procedures then the procedures can be stored on multiple servers for modularity and convenience. So by using the synchronization model presented in this paper, client and server programs can be designed with improved performance due to reduced waiting time of the process, which depends on the data generated by the other processes.

At the microkernel level, there are large numbers of communications between processes. So by using the model presented in this paper we can reduce the waiting time of any process in the kernel.

REFERENCES

[1] "Advanced Linux Programming", Mark Mitchell, Jeffrey Oldham and Alex Samel C-5(95).
[2] "Operating System Concepts 7th Edition", Silberchatz, Galvin and Gagne C-3(94).
[3] "How to make multiprocessor computer that correctly executes multiprocess programs", IEEE Transaction on Computers, Lemport, Leslie (Sep, 1979), C-28(9).
[4] Thread (Cmputing) – www.wikipedia.org
[5] Pthread (Win-32): Level of standard Conformance 2006-12-22, 2010.
[6] POSIX_Thread – www.wikipedia.org.