Study on support for distributed cooperation with Memory Based Communication Facility
メモリベース通信機構を用いた分散協調処理支援に関する研究

by

Hiroyuki KAMEZAWA

亀澤 寛之

A Thesis

Submitted to

the Graduate School of

the University of Tokyo

in Partial Fulfillment of the Requirements

for the Degree of Master of Science

in Information Science

Thesis Supervisor: Kei Hiraki　　平木 敬

Title: Professor of Information Science

February 2,1999

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Professor Kei Hiraki for his helpful suggestions and especially Takashi Matsumoto for useful suggestion for implementation of my programs and support for the MBCF and the SSS-CORE.

I would like to thank Masayoshi Nomura for his helpful advises and Niwa Junpei and Kenji Morimoto for their advises on the MBCF and the SSS-CORE. I thank all members of Hiraki laboratory.

I thank Mr.Nitroso for our friendship for these 7 years. I thank all people who gave me useful suggestions and encouraged me.

ABSTRACT

Contemporary improvement of workstation and network brings development of high-performance workstation cluster system. Viewing from the aspect of cost-performance of a system, a present workstation cluster system can provide better system than a multi-processor system. Therefore,improvement of techniques for high-performance workstation cluster is important problem today.

We have developed Memory Based Communication Facility(MBCF) as a next generation network architecture and proved using the MBCF at a workstation cluster provides high performance cluster computing. The MBCF provides secure logical address to logical address copying in a user-mode with commodity network and hardware. It provides many functions other than simple remote write/read. Memory Based Fifo (MBFifo) is function of the MBCF for providing queue service on user memory space. Memory Based Signal(MBSignal) is function of the MBCF for executing function on remote host.

In this thesis,we describe Remote Procedure Call (RPC) implemented on the MBFifo and MBSignal and evaluated performance of it. RPC is popularly used as a basic function of modern network computing system and a typical application of client-server model. RPC/MBFifo and RPC/MBSignal provides high performance RPC service with good semantics. Demonstrated performance of RPC/MBFifo and RPC/MBSignal are twice as high as that of RPC/UDP,RPC/TCP at round-trip latency.

論 文 要 旨

安価で高性能なワークステーションの登場とネットワークの性能向上によりワークステーショクラスタの性能が上昇して来た。コストパフォーマンスを考慮すれば、ワークステーションクラスタの性能はマルチプロセッサシステムのそれに匹敵すると考えられる。

我々は次世代の通信機構としてメモリベース通信 (MBCF) を開発し、メモリベース通信を用いる事でワークステーションクラスタ上のアプリケーションの性能を向上できる事を示したきた。メモリベース通信はユーザ権限によるリモートユーザ論理空間上でのメモリ操作を可能とする通信機構である。ソフトウェアのみによって実装されており、リモートメモリの読み書き以外にも多くの機能を提供している。そのうちメモリベース FIFO はユーザメモリ空間上での通信のキューイング機能を提供し、メモリベース Signal はリモートのユーザ空間上での手続き実行機能を提供する。

本論は、遠隔手続き呼び出し (RPC) のメモリベース FIFO とメモリベース Signal を用いた実装及びその性能評価について延べる。遠隔手続き呼びだしはワークステーションクラスタにおける基本的アプリケーションであり、クライアント・サーバアプリケーションの類型と言える。RPC/MBFifo と RPC/MBSignal は共に高い性能と使いよいセマンテイクスをユーザに提供する。測定された RPC/MBFifo RPC/MBSignal のラウンドリップレイテンシは UDP や TCP を用いた RPC の半分以下である。

Chapter 1

Introduction

Contemporary improvement of workstation and network brings development of high-performance workstation cluster system. A present workstation cluster system can provide better cost-performance than a multi-processor system. Therefore,improvement of techniques for high-performance workstation cluster is important problem today.

We have developed Memory Based Communication Facility(MBCF) [21][11] on a general purpose operating system "SSS-CORE"[Matsumoto94][23] [Matsumoto96][22]as a next generation network architecture on a next generation operating system(OS). The MBCF provides secure logical address to logical address copying in a user-mode with commodity network and hardware. The MBCF provides high-performance cluster computing with applications of cluster computing[9],[10]. The SSS-CORE is a scalable operating system for NUMA parallel systems. At the present,the MBCF and SSS-CORE is available on a workstation cluster consisting of SparcStation-20 workstations and 100Base-TX.

In this thesis,we describe Remote Procedure Call (RPC) implemented on the MBCF and evaluated performance of it. RPC is widely used as a basic function of modern network computing system. A programmer can call a procedure on remote host as same as to call a local procedure. RPC hides complicated part of network programming from a programmer, and provides a standardization of semantics and data format for remote execution. To improve the performance of RPC is an important problem in cluster computing. Additionally,because transaction and computing pattern of RPC is essential for all client-server applications,because the performance of RPC dominates that of client-server application on the system.

The performance of RPC depends on four elements,(1)performance of network (2) cost of sending/receiving call (3) cost of marshaling and unmarshaling of arguments and results (4) scheduling latency of invocation. We can improve (1) and (2) by using the MBCF as a network transport layer. And because the SSS-CORE provides functions for Snoopy-Spin Wait,(4) can be improved. (3) is essentially the problem of coding of RPC and ,nowadays,marshaling code for high-performance applications of RPC are optimized by hand-coding and it is out of scope of this thesis.

The MBCF provides advanced functions of remote memory management other than read and write. Memory Based Fifo (MBFifo)is a function of the MBCF,which provides first-in-first-out(fifo) packet queuing function. A user task can checking entry in MBFifo with low cost (simple local memory access). And Memory Based Signal (MBSignal) provides remote procedure invocation function on the MBCF. The invoked programs are asynchronously executed only in the scheduling periods of the target task .

We implemented RPC with MBFifo and MBSignal. RPC/MBFifo is implemented same functions of the Sun RPC[17][15] using the MBFifo. Sun RPC is a popular RPC library using UDP or TCP as its transport layer. Widely used applications such as,Network Information Service(NIS)[20] and Network File System (NFS)[6][7] are also implemented on RPC. All library functions of Sun RPC are available on RPC/MBFifo. An automatically generated code from a Sun RPC's protocol compiler,RPCGEN, is also available. A portmap service for RPC/MBFifo returns a name of service task in spite of port number.

RPC/MBSignal is implemented as a library routine using the MBSignal. RPC/MBSignal is not full-compatible with Sun RPC, RPCGEN is not available, but RPC/MBFifo uses basic ideas of Sun RPC, and its interface is similar to that of Sun RPC. A portmap service for RPC/MBFifo also returns a name of service task in spite of port number.

In this thesis,we describe the MBCF precisely in Chapter 3. In chapter 2,we describe RPC and introduce some preceding RPCs. Chapter 4shows our implementation of RPC/MBCF and chapter5 shows performance evaluation of RPC/MBCF. chapter6 is a conclusion of this thesis.

Chapter 2

Remote Procedure Calls

2.1 Remote Procedure Call

Remote Procedure Call (RPC) is a software library for remote execution. RPC was first introduced by Birrel and Nelson in 1984[2]. With RPC,user can call a procedure on remote host in the same way of calling a local procedure. A RPC library hides message packing/unpacking and transferring from user. In general RPC client hides these processing form user.

1. User calls a local procedure with server name. This procedure call RPC library routine. Arguments are passed by reference.
2. RPC library packs passed arguments into packet, send it to a indicated server.
3. RPC library waits for reply packet from server,and receive it. Client timeout routine may be managed by RPC library.
4. RPC library unpack reply packet,and write results into indicated memory address.

In addition,RPC library provides users with a protocol compiler. Protocol compiler makes it easy to use RPC. This software automatically creates message packing/unpacking routine,packet message formats,and so on. RPC freed users from troublesome programming of “usually” used transaction code and offered standardized remote execution protocol.

Studies on RPC was hot from 1980s to early 1990s. These studies focused upon speed-up of RPC. RPCs are classified into “using IP(Internet Protocol)” and “using RPC dedicated protocol(without IP)”.

A merit of implementing RPC with IP is (1)IP is supported by almost all operating system and developer can use “safe” IP code. Developer can reduce cost of implementation. (2) IP traffic can pass general network hardware,rooter,switch and so on. Usually, User Datagram Protocol(UDP) is used as a connection-less protocols on IP. Some uses raw IP as a communication layer with original protocol. Others uses Transfer Control Protocol(TCP) as a connection oriented protocol layer of RPC,RPC fails only with fatal network error, but this implementation has great overhead.

Using completely RPC dedicated protocol is more effective and faster than using IP. Implementation is more difficult,but RPC dedicated protocols has more suitable features of (1) reducing copy operation of packet (with no protocol layer) (2) guaranteed data transferring (3) more suitable packet header.

RPCs are also classified into “blocking RPC” and “non-blocking RPC”. With Blocking RPC,caller is blocked until response from server is received. If RPC relies on unreliable protocol which doesn’t guarantee packet-arrival,blocking RPC is popularly used for transaction’s timeout processing . Non-Blocking RPC is further classified into “Non-Blocking without return value ” and “Non-Blocking with return value”. Reliable communication protocol is suitable for non-Blocking with return value RPC.

Semantics of RPC is how packet-loss of RPC influence the execution of RPC. In general,a reliable transport layer provides “one result per one request”. semantics.With an unreliable transport layer ,a re-transmission of request or result packet occurs at packet-loss.Therefore,in this time, semantics of RPC is ”one result per a several request” or “several result per a several request”. Of course,most useful semantics is “one result per one request” semantics and this semantics requires reliable transport protocol.

In chapter3,we describe the Memory Based Communication Facility(MBCF). The MBCF guarantees arrivals of packets and keeps order of them between one node and the other node.This characteristic of the MBCF provides suitable semantics “one result per one request” and makes implement of Non-Blocking RPC very easy.

2.2 Related Works

In this section ,we introduce some studies on RPC and Sun RPC as most standard RPC library of today.

2.2.1 Xerox Cedar RPC

Xerox Cedar RPC is a one of earliest implimentaion of RPC. It was developed by Birrel and Nelson. Their paper shows good RPC model and implementation options and their decisions. It was cited from many papers and regarded as a landmark of RPC implementation.

Cedar RPC was implemented as a remote procedure call package within Cedar programming environment.Their aims on this project are (1) to make distributed computation easy. (2) to make communication highly efficient. (3) to make the semantics of the RPC package as powerful as possible. (4) to provide secure communication of RPC. Also they introduced "stub" structure which is well-known today. (see 2.1) User calls user-stub and passes arguments to it. User-stub packs arguments into packet and passes it to user RPC Runtime. User RPC Runtime transferring packets to server RPC Runtime.server-stub unpacking arguments and pass them to server program.In this case, RPC Runtime is provided by the Cedar system and stubs are automatically generated by a program called *Lupine*.Only user have to do is write "server" and "client" program.(this realizes aim of (1)).

For the aim of (2) they implement RPC dedicated protocol. Features of RPC dedicated protocol are to improve round-trip latency not data throughput. For this purpose,they omits expensive connection handshake,they makes connections between client machine Runtime and server machine Runtime.This omits connection establishment and close packet for most of connection oriented transport protocol. If a runtime of server host is alive,there is no timeout mechanism for remote procedure call,that is,only if networks or Runtime, server program is down, RPC is aborted. This is a semantics of their RPC (3).

For aim of (4),they implemented encrypted RPC using Grapevine[12] distributed database as authentication server.

2.2.2 Firefly RPC

Firefly RPC was implemented on Firefly multiprocessor system. M.D.Schroeder and M. Burrows[14] shows analysis of implimentaion to account precisely for all measured latency.(See Table2.1 and Table2.2)

Firefly multiprocessor system consisted of 5 Micro VAX II processors and a DEQNA Ethernet Controller.

RPC on Firefly system had these characteristics.

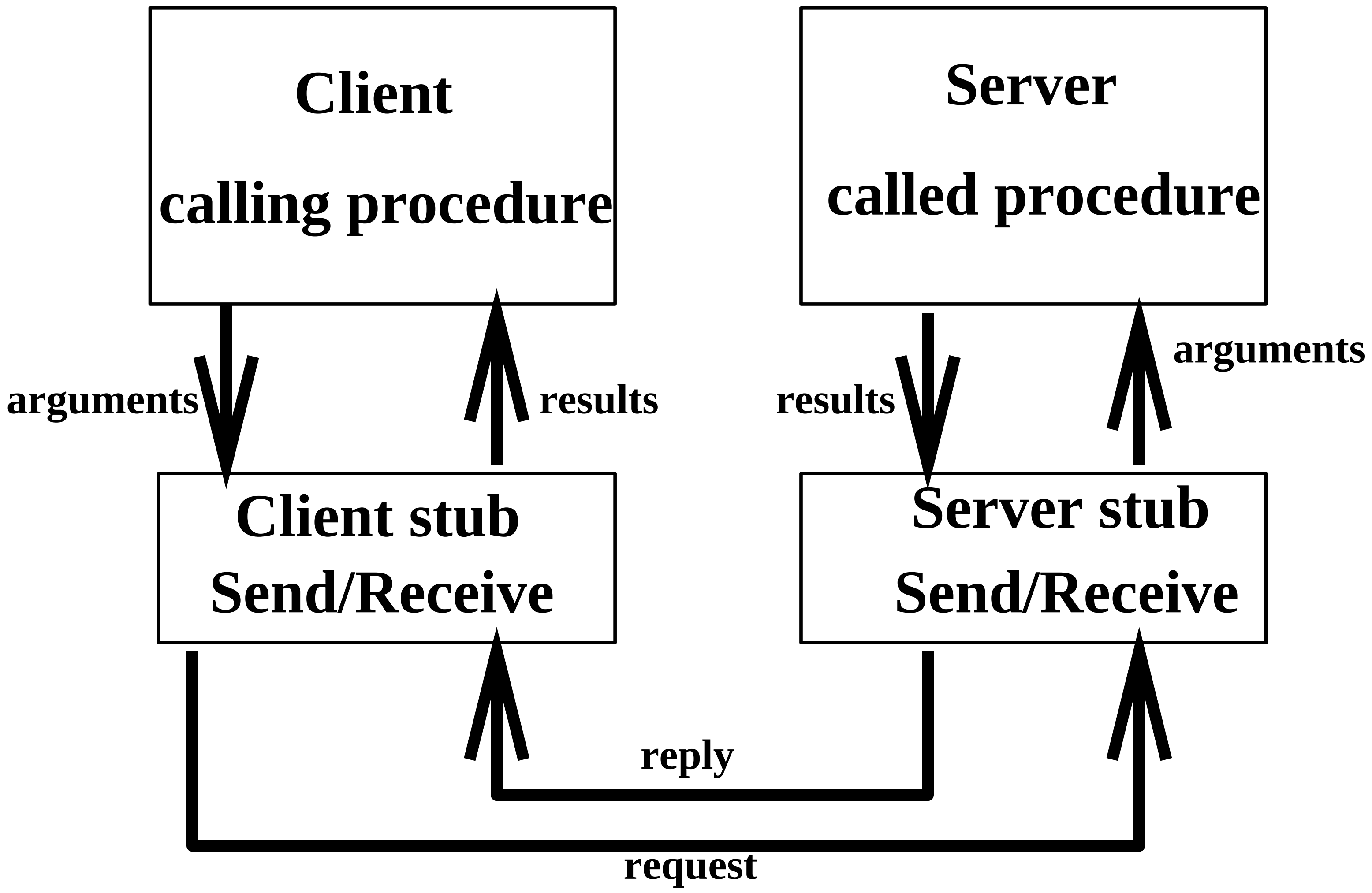

Figure 2.1: Stub structure of RPC

Machine	Procedure	μs
hline Caller	Calling program(loop to repeat call)	16
	Calling stub(call and return)	90
	Starter	128
	Transporter(send call packet)	27
Server	Receiver (receive call packet)	158
	Server stub (call and return)	68
	Null server procedure	10
	Receiver (send result pakcet)	27
Caller	Transporter(receive result packet)	49
	Ender	33
Total		606

Table 2.1: Latency of Stubs and RPC Run Times of Firefly RPC

Procedure	Action	μs
Null()	Caller,Server,stubs,RPC runtime	606
	Send+Receive 74byte call packet	954
	Send+Receive 74byte result packet	954
	Total	2514
MaxResult(b)	Caller,Server,stubs,RPC runtime	606
	marshal 1440byte result packet	550
	Send+Receive 74byte call packet	954
	Send+Receive 1514byte result packet	4414
	Total	6524

Table 2.2: Calculation of Latency of RPC to Null() and MaxResults(b) on Firefly RPC

- Firefly RPC was implemented on UDP/IP.
- Stubs were automatically generated from a Modula2+ definition module.
- Packet buffers were mapped in shared address space between user process and network driver routine reside in VAX I/O space.
- RPC demultiplexing routine (awaking server routine) was implemented in Ethernet handler routine.

Because shared mapped I/O space omitted copying between user and kernel, copying cost of firefly RPC was very low.In their measurements, costly step was transferring on Ethernet,marshaling/unmarshaling of arguments, calculation of UDP checksum.

After analyzing Firefly RPC,they advised on RPC implementation that (1) use more fast hardware (2) omit UDP checksum (3) omit layering on IP and UDP (4) use busy wait.

With memory-mapping and direct function execution from device driver, Firefly RPC got very low latency. But its implementation was not for User-to-User RPC on time-sharing system.single-user but for single-user system or kernel to kernel RPC.

2.2.3 Sun ONC/RPC

Sun ONC/RPC was developed by Sun Microsystems as part of the Open Network Computing. Sun RPC is now one of popularly used RPC package and its distribution can be got free. Sun RPC protocol is used as a fundamental communication layer in NFS/NIS.

Sun RPC provides both UDP and TCP for transport communication.ONC eXternal Data Representation(XDR) standard package is used for Data marshaling/unmarshaling protocol. Sun RPC provides blocking RPC, batching RPC, broadcast RPC.Batching RPC is a kind of non-blocking RPC.Its calls expect no return and the last call must be a normal blocking RPC call in order to flush all the calls.When broadcast RPC was called, only servers respond which successfully completed a request.

Sun RPC provides two types of interface for application programmers. One is RPC specification language (RPCL). RPCL is an extension of XDR and stub generator (RPCGEN) creates stub code from RPC protocol definition written by RPCL.(See Figure2.2) Second is RPC library routine to use RPC directly. RPC library routine consists of three layers, from easy use layer to difficult and powerful layer. Figure2.3 shows the highest interface "*call_rpc*()". A stub code automatically

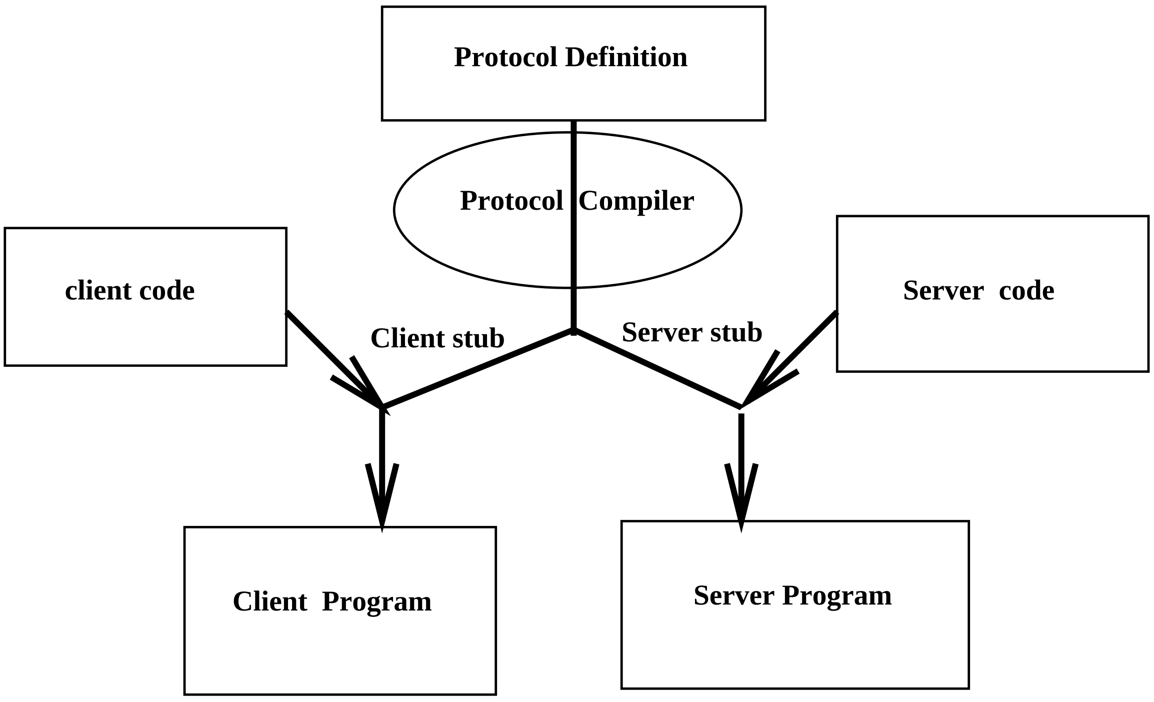

Figure 2.2: Application development with RPCGEN

1. A programmer writes (1) protocol definition (2) client code (3) server code
2. RPCGEN generates from protocol definition (1) client stub (2) server stub (3) XDR routine (4) header file
3. A programmer compile and link them.

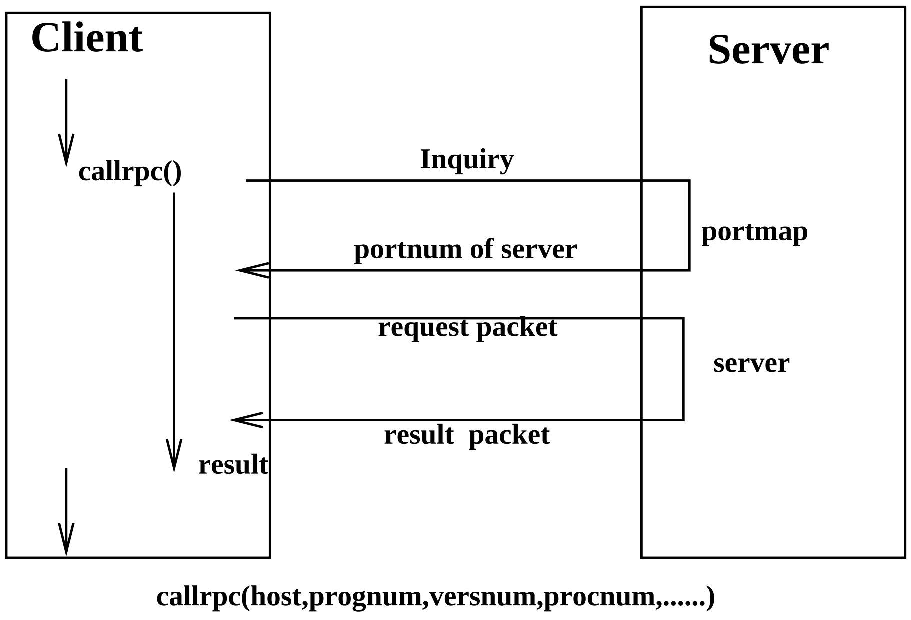

Figure 2.3: callrpc()

$callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)$

- host −− name of server host.
- prognum −− service's program number
- versnum −− service's version number
- procnum −− requesting procedure
- inproc −− input marshaling XDR routine
- in −− pointer to input argument
- outproc −− result unmarshaling XDR routine
- out −− pointer to result buffer

generated from RPCL is very easy to maintain where stub is actually hidden.Some application programmer uses lower layer library routine and powerful ones write codes by hand coding from raw Sun RPC protocol to omit overheads of calling library.

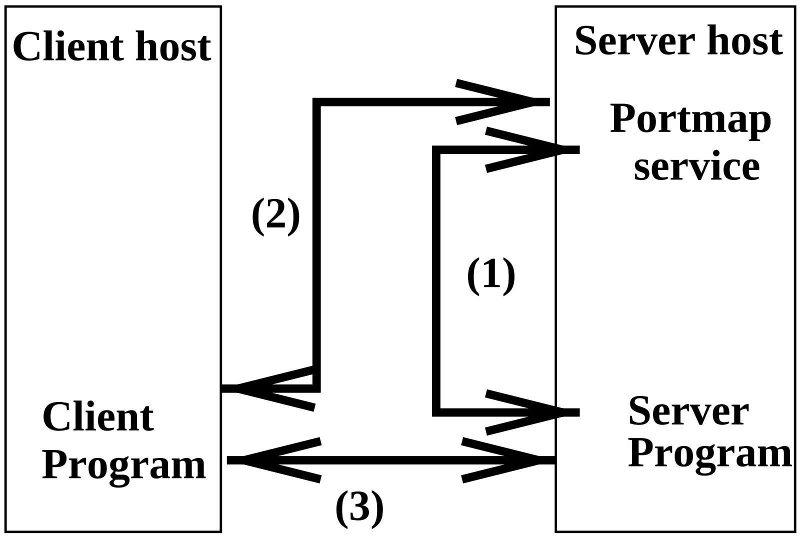

Figure 2.4: Portmapper of Sun RPC

1. A server program register itself to portmap's RPC table.
2. A client asks portmap what ports a service program is waiting at. A portmap replies the registered entry
3. A client send request to a server program.

An application of Sun RPC finds its server by accessing Portmapper. Portmapper is also written by RPC. When Portmapper receives RPC request from client application,it searches its application table which a server program register itself to ,and then it returns server's port number as a response of RPC request.

Sun RPC is very portable and flexible RPC.The major application developed by Sun using Sun RPC is called Network File System(NFS). NFS is very popular application today and is ported to many plathomes. But using TCP/UDP/IP for transport layer of NFS limits performance of NFS.Actually,important application is usually hand coded from Sun RPC protocol to gain some speed-up.

Chapter 3

Memory Base Communication Facility

3.1 Outline of Memory Based Communication Facility

Memory Based Communication Facility(MBCF) is developed as a software-only communication system which provides protected and virtualized high-speed user-level communication and synchronization. The MBCF was developed for workstation cluster and distributed memory multiprocessors without hardware Distributed Shared Memory (DSM) mechanisms.

The MBCF is a software-only realization of logical address to logical address data transferring mechanism.It realizes high-performance data transfer making use of hardware mechanism of modern processor's memory management unit.

To achieve high-speed user-level communication,the MBCF was designed to be (1) to allow senders to specify where to write packets into receiver's address space. (2) to reduce calling cost of MBCF function with MBCF dedicated systemcalls (3) to give users an opportunity for optimizing the number of packets. (4) to allow users to schedule themselves for making timely transaction strategy with opened transaction information. (Figure 3.1) Of course,code of the MBCF interrupt routine is as optimized as possible.

The MBCF is now implemented on SSS-CORE operating system for SparcStation-20 work station cluster using Fast Ethernet(100BASET-X) and Ethernet(10BASE-T).[21]

- Axil 320 model 8.1.1 (Sun SparcStation 20 compatible,85MHz,SuperSPARC $\times$ 1)
- Sun Microsystems Fast Ethernet SBus Adapter 2.0
- 3Com Hub/TP100

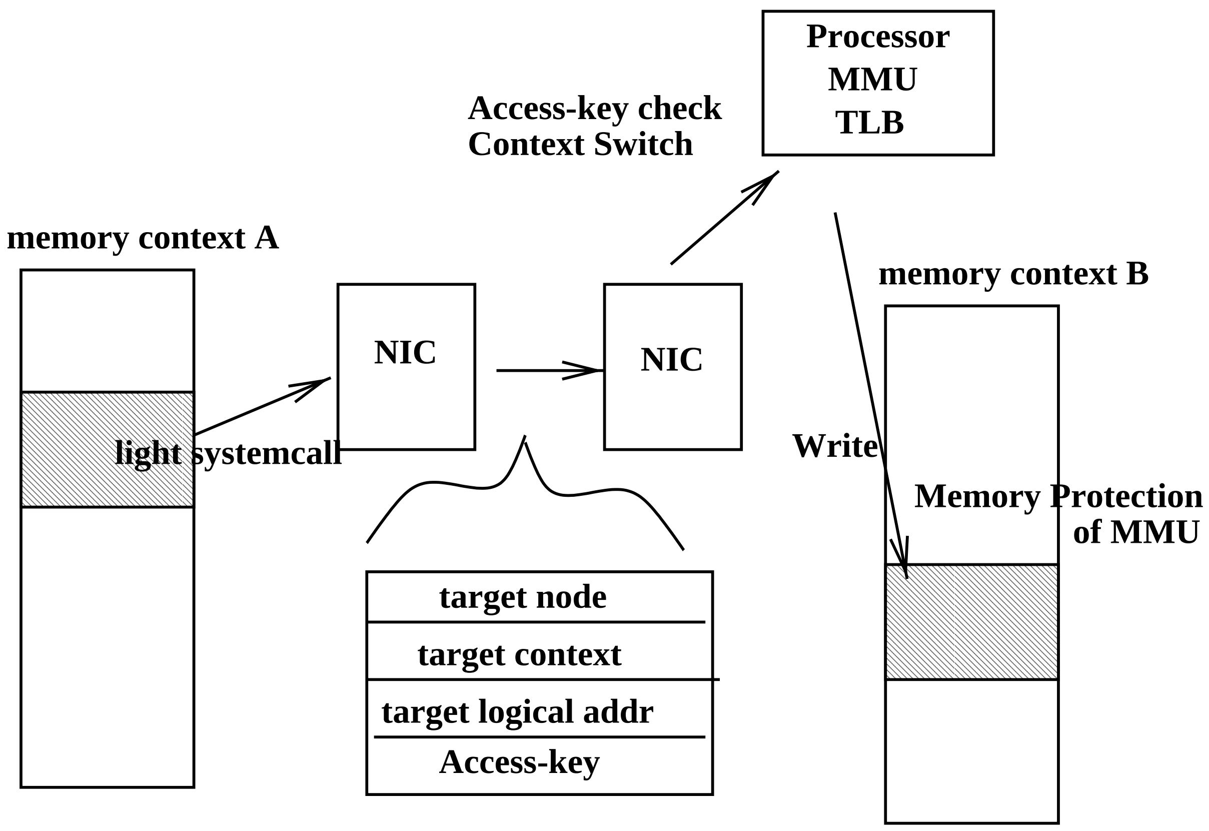

Figure 3.1: the Memory Based Communication Facility

Table 3.1: Peak bandwidthes of the MBCF/100BASE-TX

data size (byte)	4	16	64	256	1024	1408
100BASE-TX (Mbyte/s)	0.29	1.06	4.03	8.28	10.86	11.24

Table 3.2: Round-trip latency of the MBCF/100Base-TX

data size (byte)	4	16	64	256	1024
round-trip latency (μs)	49	54	60.5	88	200

An operating system of each nodes is SSS-CORE.

Table 3.1 shows peak bandwidth of the MBCF/100BASE-TX. This table shows peak bandwidth of MBCF_WRITE of Various length of data. The measurement method is that a requester sends MBCF_WRITE packet to a fixed target without checking any acknowledgements except for an acknowledgement of every 16 requests.These checkings are inserted for avoidance of saturation of the Ethernet. (Note:data length in table are not including length of MBCF/Ether header field.)

Table 3.2 shows round trip latency of the MBCF/100Base-TX. The measurement method of round-trip latency is that a requester send MBCF_WRITE with acknowledge requesting option and difference between time of calling a systemcall for the MBCF and time of confirming its acknowledge ment.

3.1.1 Flexibility of Direct Memory Access

The MBCF's policy of direct memory access has advantage in flexibility and simplicity over conventional message-passing-type system interfaces which has excessive receive buffer abstraction. Message-passing-type interface provides users with "send" and "receive" interface to access remote node. With message-passing-type interface,sender can set packet's target only to a abstraction of kernel's message-buffer which receiver process belongs to. This typical policy of message-passing-style limits flexibility of communication ,that is

- "Send" and "Receive" interfaces limit target logical address to kernel's a few implicit message-buffer address.
- "Send" and "Receive" interfaces limit type of requests only one, remote-memory-write.

- Correspondences between "Send" and "Receive" is essential for message-passing-type interfaces.And this correspondence This limits a flexibility of "send" and "receive"

With the MBCF,which provides direct remote memory access style, first shortcomings are eliminated.Because of specifying remote memory address, With logical address information,the MBCF provides more complex packet handling method than simple remote memory write operation.(READ,QUEUE,Signal,Fetch and Add,Swap etc ....) Address information is very important to implement a little complex packet handling method.

The MBCF realize user level direct remote memory access by allowing the interrupt routine(kernel mode) to execute direct access to user space. And recent high-end processorslike SuperSPARC, UltraSPARC stands for interrupt routine like that with the following mechanisms.

- the processor in the kernel-mode can perform memory-accesses with user-privilege without paying penalties
- Many pages from various task-spaces can exist at the same time in processor's TLB.
- Changing the current context is inexpensive and costs only one instruction and a few clocks.

These characteristics of processors allows interrupt routines to executer direct access to user-space with low penalties.

3.1.2 Specifying remote memory address

Conventional user-level communication interfaces, such as Unix's socket(TCP/IP,UDP/IP) and MPI,have a characteristic that send operations are dealt as remote write operation to specific message buffer in kernel space. This makes two shortcomings for large overheads.

(1) Sender cannot specify receiver's a receive buffer in detail. This requires senders to packing information into packet and receivers to unpacking packets into suitable (separated) address space.

(2) Users has to copy received data from kernel space to user space.

With the MBCF, sender can specify receiver's buffer address by virtual global address. It solves those two shortcomings.(For application which wants to communicate by FIFO, the MBCF provides Memory based FIFO.This function is introduced later.) The MBCF's virtual global address is consists of a logical-task-ID and a logical address in the target logical-task as we write the combination as (Ltask:Laddr). The task in the MBCF system is a abstraction of processor's activity and has its own memory-space.In the MBCF system,task is specified by a the combination of physical-node-ID and physical-task-ID,to say, logical task(Ltask) is represented as the combination of them,(Pnode:Ptask). The Operating System (OS) for MBCF system maintains a logical task translation table for each task.

3.1.3 Interface for the MBCF

A light-weight systemcall is a systemcall omitting operations for keeping register context,to save and restore registers,and so on.Therefore, no context-switches can take place during light-weight systemcall. A light-weight systemcall call reduces cost of calling kernel-level routine.

Systemcalls for MBCF is implemented as light-weight systemcalls.

When user-level application sends a MBCF packet, it makes a body part of MBCF-request packet and sets a parameter and calls a light-weight systemcall with to send a packet. User application can easily make short packets combine into one large packet through this application interface.

3.1.4 Protection and Security of the MBCF

For protection and security of the MBCF, the MBCF exploits page-aliases mechanism and a simple access-key system and not uses heavy capability checking codes. To say, only memory areas mapped to the target task can be accessed through the MBCF and the task's memory area is protected from MBCF with a unique access-key. This is very smart protection system exploits hardware memory management mechanisms. The Requester of MBCF has to know the target's taskname and its access-key. Usually,an access-key is exchanged during creating logical-task table entries. Problem for access-key exchanging and showing all of task's memory area is discussed later section,Client-Server with the MBCF.

3.2 Functions of the MBCF

3.2.1 Overview

The MBCF provides a variety of functions elsewhere a simple remote memory write. Functions of the MBCF are listed in the following list.

WRITE_TASKQUE Every task in the MBCF system has a unique small queue.This queue is called taskque. The task which attempt to establish a connection with another task calls "send-taskque" systemcall.The send-taskque systemcall register a target task into the task's Ltask table and send a WRITE_WRITE packet to the target task. The MBCF command "WRITE_TASKQUE" enqueue an packet to the target task's taskque. When WRITE_TASKQUE command successfully completed, the sender task is registered into the target task's Ltask table.The target task can dequeue an item from the taskque by calling "read-taskque"systemcall. At the same time, the target task also can get an access-key of the requester with read-taskque systemcall.

WRITE The MBCF command "WRITE" writes a body of a packet into specified Ltask's memory address.

READ The MBCF command "READ" copies data of specified memory area of the target task into the specified local task's memory area.

Memory Based Fifo(MBFifo) The FIFO interface of the MBCF. A task creates memory area for MBCF_FIFO in its own address space and this FIFO is called Memory Based Fifo. Enqueuing to fifo is executed by the MBCF command and dequeuing from fifo is executed by the light-weight system call.This dequeue is only copying in the task's address space. This function is precisely introduced in later section.

Memory Based Signal(MBSignal) MBSignal is a remote invocation method of the MBCF. The client task which attempt to do remote invocation sends a MBCF_SIGNAL packet and a registered procedure is called with the privilege of the target task asynchronously. This function is precisely introduced in later section.

Atomic Operations Atomic operations such as swap,fetch and add,compare and swap etc... are atomically executed in interrupt routine.

Common characteristics of all MBCF command is

- Because the MBCF leaves the packet fragmentation in user's hands,a data of packet sticking out the range of MBCF_MAX_DATA_LEN is discarded.
- Short packets of the MBCF can be packed into one packet with simple setting of pointer.These short packets have to be packets of a same function.

- A task can set an address of acknowledge flag in all of the MBCF request(NULL is available). When a MBCF request is successfully completed,the target task of it writes an acknowledge into specified acknowledge flag address. This acknowledge is automatically generated and returned. When acknowledge flag address is NULL,an acknowledge is not generated.
- An access-key of a connection requester is handed to the target task during "read-taskque" systemcall.On the other hand, the target task's access-key is not implicitly informed ,the connection requester is defenseless to the target task.

3.2.2 Memory Based Fifo

The MBCF is designed to support distributed shared memory system but is supports a message-passing-style with the MBCF function "Memory Based Fifo (MBFifo)".

A task creates a buffer for queue and structure for information of queue in its own address space. Therefore,there is no limitation of the number of MBFifo. The MBCF request packet for enqueuing sets its target address to the address of information structure of the MBFifo.An interrupt routine for the MBCF reads the MBFifo information structure (in user area) and enqueue a body of the packet to the buffer specified by the information structure.(Figure3.2) Dequeuing is executed with a light-weight systemcall.Because a buffer and an information structure for queue exist in task address space, What to be done in this systemcall is simply local memory copy and updating an information structure.

Because of exclusive enqueuing/dequeuing operation, dequeue from MBFifo need to be executed with a light-weight systemcall.But probing of Fifo is feasible without calling systemcall which doesn't update any information about queue.Because an information structure of Fifo exists in the task's own address space, the task can check easily whether the Fifo is empty or not.This probing function enable the task to do Snoopy-Spin-Wait(SS-Wait), that is the task can determine what to do (sleep/execute and so on...) with an information of waiting status of queue.

3.2.3 Memory Based Signal

Memory Based Signal(MBSignal) is a function of the MBCF for remote invocation of a user specified procedure with the privilege of the target task It also provides users with the method of asynchronous communication in the MBCF scheme. The Memory-Based Signal is an inter-node

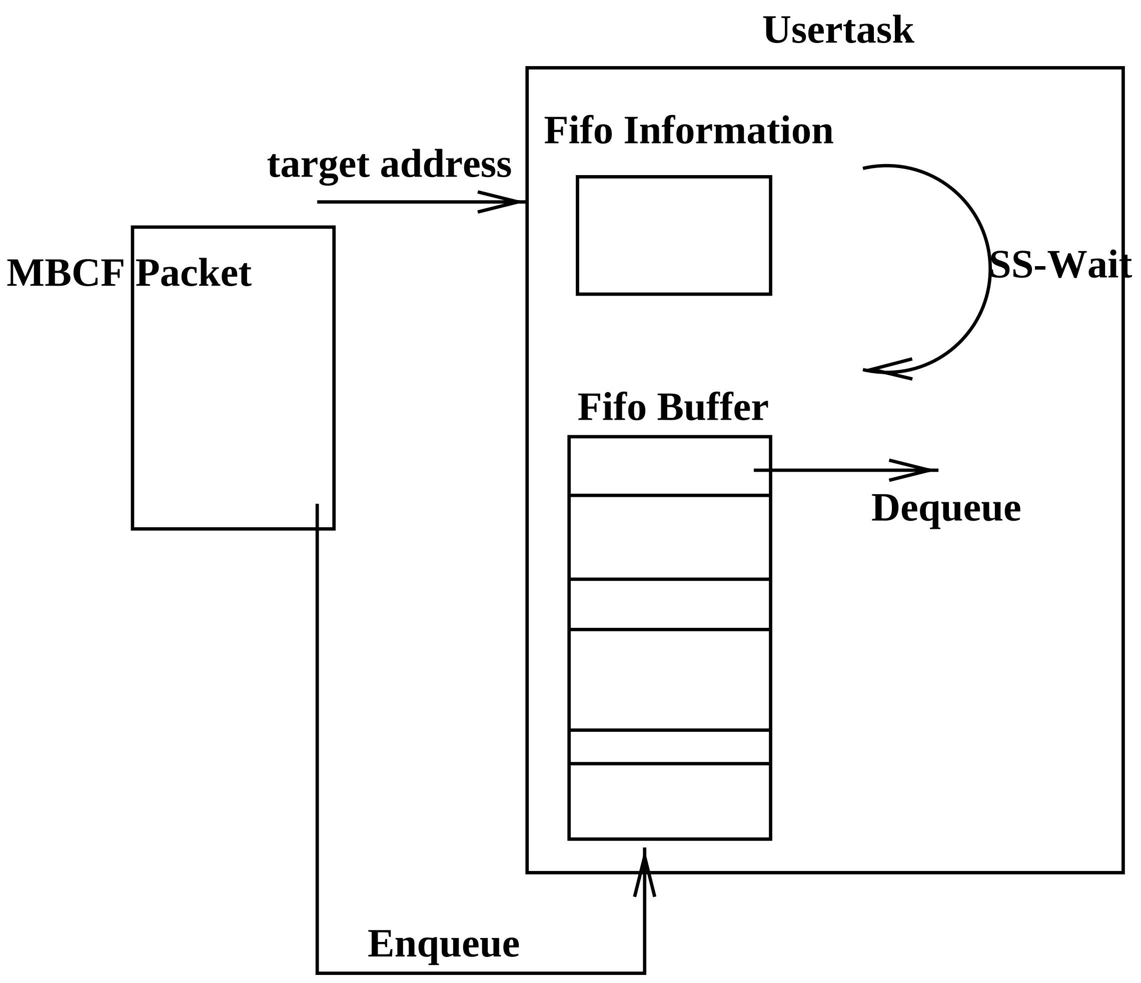

Figure 3.2: MBFifo

and memory-based extension of UNIX signals and it is promising primitive for making high-speed Remote Procedure Call and/or Remote Method Invocation.

The invocation mechanism of the MBSignal is similar to that of UNIX's signals, and invoked programs are executed only in the scheduling periods of the target task. The server task for remote

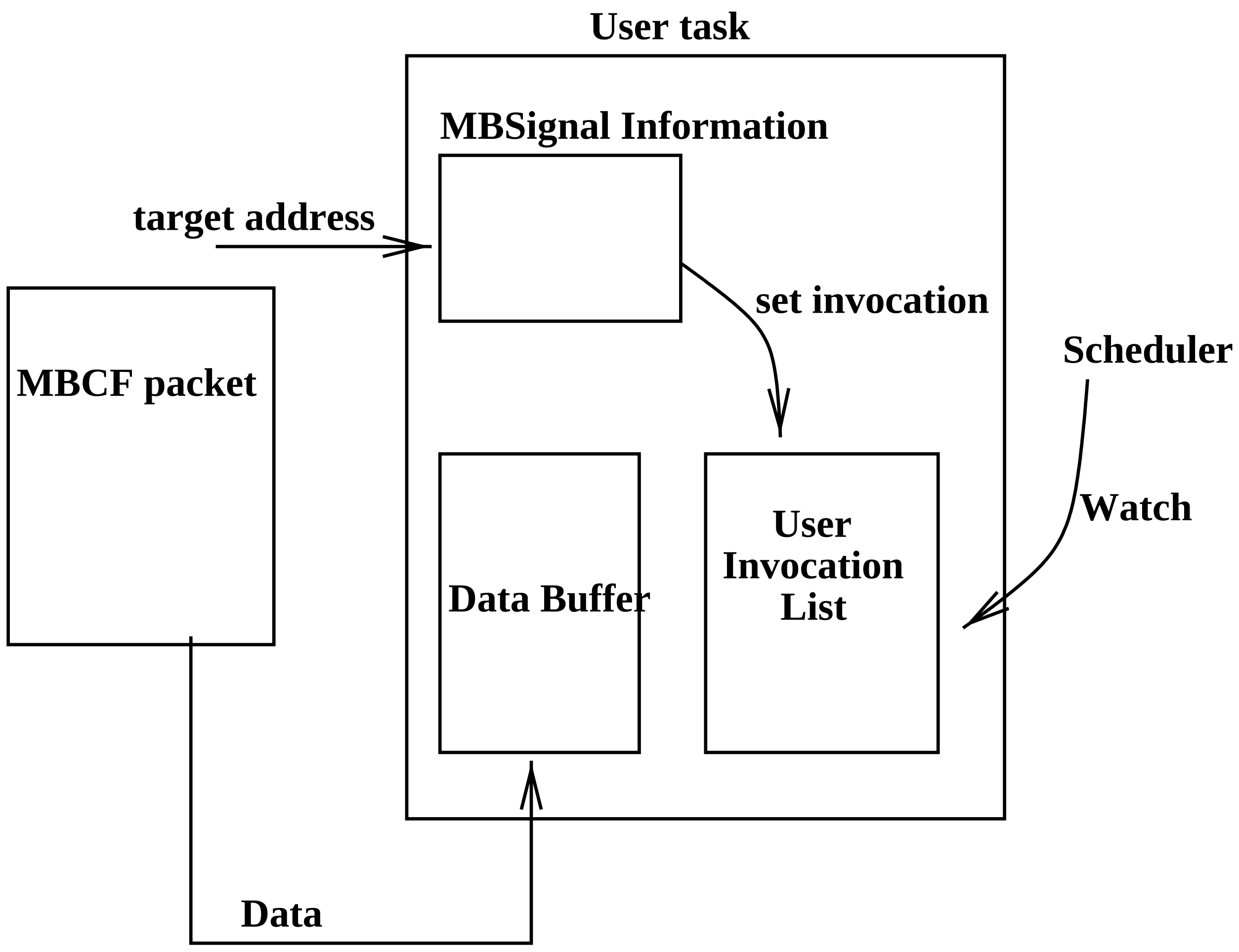

Figure 3.3: Memory Based Signal

invocation prepare a structure for MBSignal. This structure includes environment for invocation of registered procedure and memory address of registered procedure.address of a data buffer. A MBCF_SIGNAL packet's target address is set to an address of the structure. (Figure 3.3)

The task can also carry a data(arguments) in body part of packet with MBSignal to the target procedure. The prepared information structure keeps information for a data buffer and the server can use a simple buffer or MBFifo for a data buffer.

First, the MBSignal packet is received,the information structure is accessed according to the address specified by the packet.In this turn, an interrupt routine checks invocation-type of MBSignal,an entry of the information structure.

Invocation-types are classified into (1) invocation according to User-Level-Scheduler(ULS) (2) invocation of Destination-Specified Program(DSP) (3) invocation of Source-Specified-Program(SSP). ULS is the special routine for scheduling threads of the target task. At most one ULS per task is registered into OS by the target task itself. DSP is an user-level procedure of the target task.Address of DSP is registered in the information structure. SSP is an user-level procedure of the target task.Address of SSP is specified in the MBSignal packet. The requester can specify address of an arbitrary procedure in the target task. Therefore,this is very powerful but very dangerous invocation type.

Second, the interrupt routine checks a parameter of “frequency” in the information structure.”Frequency” parameter specifies a frequency of invocation. MBSignal provides following frequency parameter types.

Every Time Block If invocation-flag is 0,interrupt routine invokes specified procedure and set invocation-flag to 1.If invocation-flag is 1,no invocation occurs. This flag can be clearly reseted by the user task.

Every Time NonBlock Regardless of invocation-flag a specified procedure is invoked.

First Only NonBlock If invocation-flag is 0,interrupt routine invokes specified procedure and set invocation-flag to 1. If invocation-flag is 1,interrupt routine only pile up the request argument in the buffer,no invocation occurs. User procedure of executing MBSignal can check its argument buffer, and,if necessary,take new arguments from buffer and restart its execution. After all of requests are completed,user procedure set invocation-flag to 0 and exit MBSignal procedure.

Third, the interrupt routine store data for a target procedure into specified buffer.Types of buffer is two,(1) a simple buffer: arguments are written with MBCF_WRITE (2) MBFifo.

Finally,the interrupt routine set execution request of MBSignal into a invocation queue of the target task.When the target task is scheduled,this MBSignal procedure is invoked in the privilege of the target task within the scheduling period of it.This invocation occurs asynchronously.Even if the target task is sleep state, the invocation is scheduled and occurs.

Chapter 4

RPC/MBCF

The MBCF has desirable characteristics for RPC. First,it guarantees an arrival of packet and it keeps an arrival order of packet on a connection.Therefore Semantics of RPC on the MBCF(RPC/MBCF) is naturally "one result per one request". Second,the MBCF is low round-trip latency protocol and the most important parameter of RPC is round-trip latency. Third,because the MBCF provides a logical address to logical address memory copying service,the server procedure of RPC can easily access client task's memory space.This expands latitude of a service procedure. Finally,Memory Based Signal is a suitable function to implement RPC.

The MBCF is originally designed to support distributed shared memory applications or distributed parallel ones.But RPC is a typical application of using client-server model. In general,the RPC server has to be protected from an access from any client.Because the MBCF enables a client task to access an arbitrary memory address of a server task,some protection method has to be installed in the server task.

We implement RPC/MBFifo and RPC/MBSignal. RPC/MBFifo is based on SUN RPC library and RPC/MBSignal uses original implementation.

In this chapter we show

1. general protection method of a server from a client.
2. implimentaion of RPC/MBFifo
3. implementation of RPC/MBSignal
4. performance evaluation of RPC/MBCF

4.1 Implementation of client-server with the MBCF

In this section,we describe the protection method for client-server programming with the MBCF.

4.1.1 The practical side of client-server implementation with the MBCF

If a server can assume that all of clients are reliable,the later four methods are unnecessary and its implementation is very simple.It is unnecessary to protect server task's memory space and its clients can directly access to it.

On the other hand, if clients are unreliable,the server have to protect its memory space from invalid access from its clients.

The most important protection is strict authentication of clients and careful provision of server's access-key.If a client is judged reliable,the focus of protection is how to protect server's memory space from its clients.

There is two way to protect a server from a client. One is not to inform a client of server's access-key.With this method, a server is completely protected from its clients.Another is to divide server's memory space into memory area for interface to the clients and server's important data or code memory area. The following sections introduce these protection methods precisely with a typical four methods. In general,combination of these four method is suitable for implementation of client-server with the MBCF.

The following sequence is a general sequence of starting transaction with the MBCF for client-server transaction.(Figure 4.1)

1. A client task set its own access-key.
2. The client call "send-taskque" systemcall to establish connection with a server task. In this systemcall,the server task is registered into the client task's logical task table and WRITE_TASKQUE command packet is send to the target.The client task can fill the body of WRITE_TASKQUE command packet arbitrary.
3. The server call "read-taskque" systemcall to read entry from its taskque. In this systemcall,the client task (the sender of WRITE_TASKQUE) is registered into the server's logical task table and the access-key of the client and the body of the packet is passed to the server task.

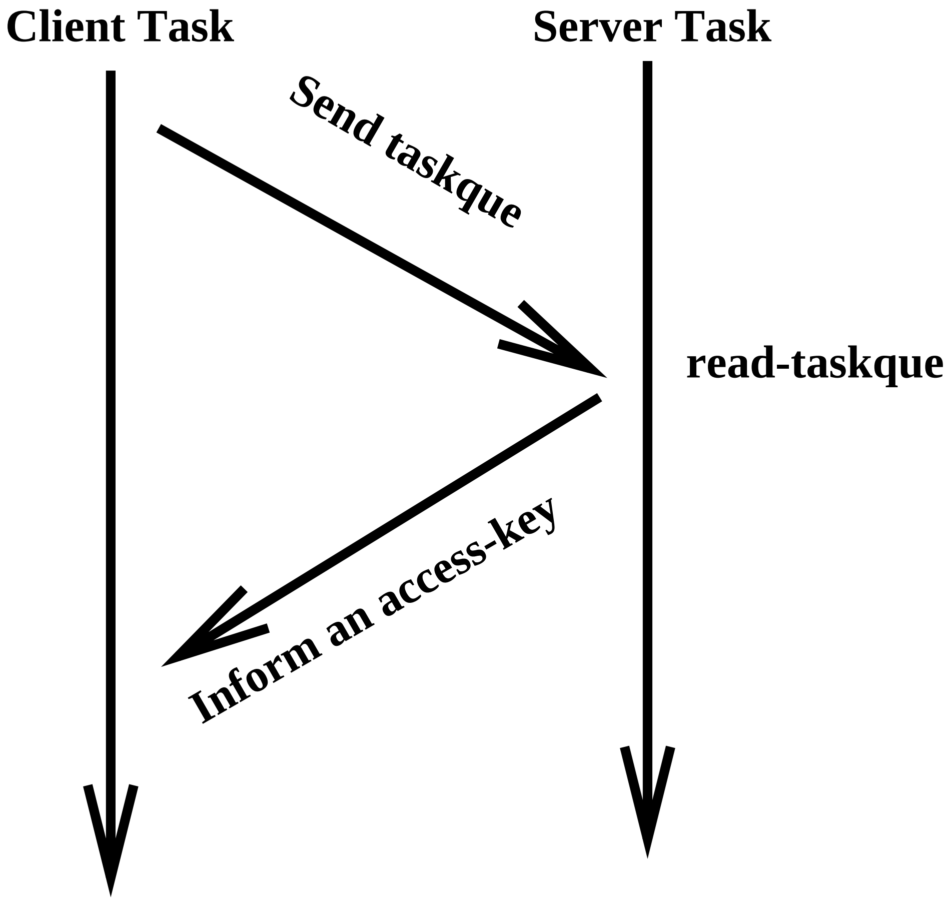

Figure 4.1: Connection Establishment Sequence of the MBCF

4. A connection between the server and the client is established. At this point, the server knows “access-key” of the client and can access the client with any command of the MBCF. Until the server informs its access-key to the client,the client cannot use the MBCF command anything but calling “send-taskque”.When the server doesn’t allow its clients to access with the MBCF,it doesn’t need to set its own access-key.

A transaction style between a server and a client is roughly classified into (1) a server gives its access-key to its client. (2) a server doesn’t give its access-key to its client.

When a server informs its access-key to its client,it need to test its client carefully.A server must refuse connections with unsuitable clients.

When a server doesn’t inform the client of its access-key,request data from a client is small enough to fit in a WRITE_TASKQUE packet or a server bothers to read its client task’s memory. In this case, overhead of “read-taskque” and passive calling of MBCF_READ cannot be bypassed. We introduce four styles of client-server transaction with the MBCF and discuss characteristics of each style in following section.(Figure 4.2)

aggressive use of taskque A server doesn’t inform the client of its access-key . All of the request is transfered within a WRITE_TASKQUE, connection-establish packet.

passive-read A server doesn’t inform the clients of its access-key . A client task sends addresses of its requests within a WRITE_TASKQUE and a server read a client task’s request buffer.

dummy task with shared memory A server prepares a dummy task for receiving packets from client.A shared memory is mapped between a server task and its dummy task and it is used for request buffer of the server.

interface task creation A server prepares a dummy task to deal with request.(we call it “interface task.”) In this case,a dummy task is a simple interface to the client.

4.1.2 Aggressive use of taskque

When a client sends a connection-establish packet,”WRITE_TASKQUE”,its request is packed in the same packet.A server read the request from its taskque. After the server deals with the request ,it ,if necessary, writes back result to the client task and eliminate the connection. Advantages of this approach is

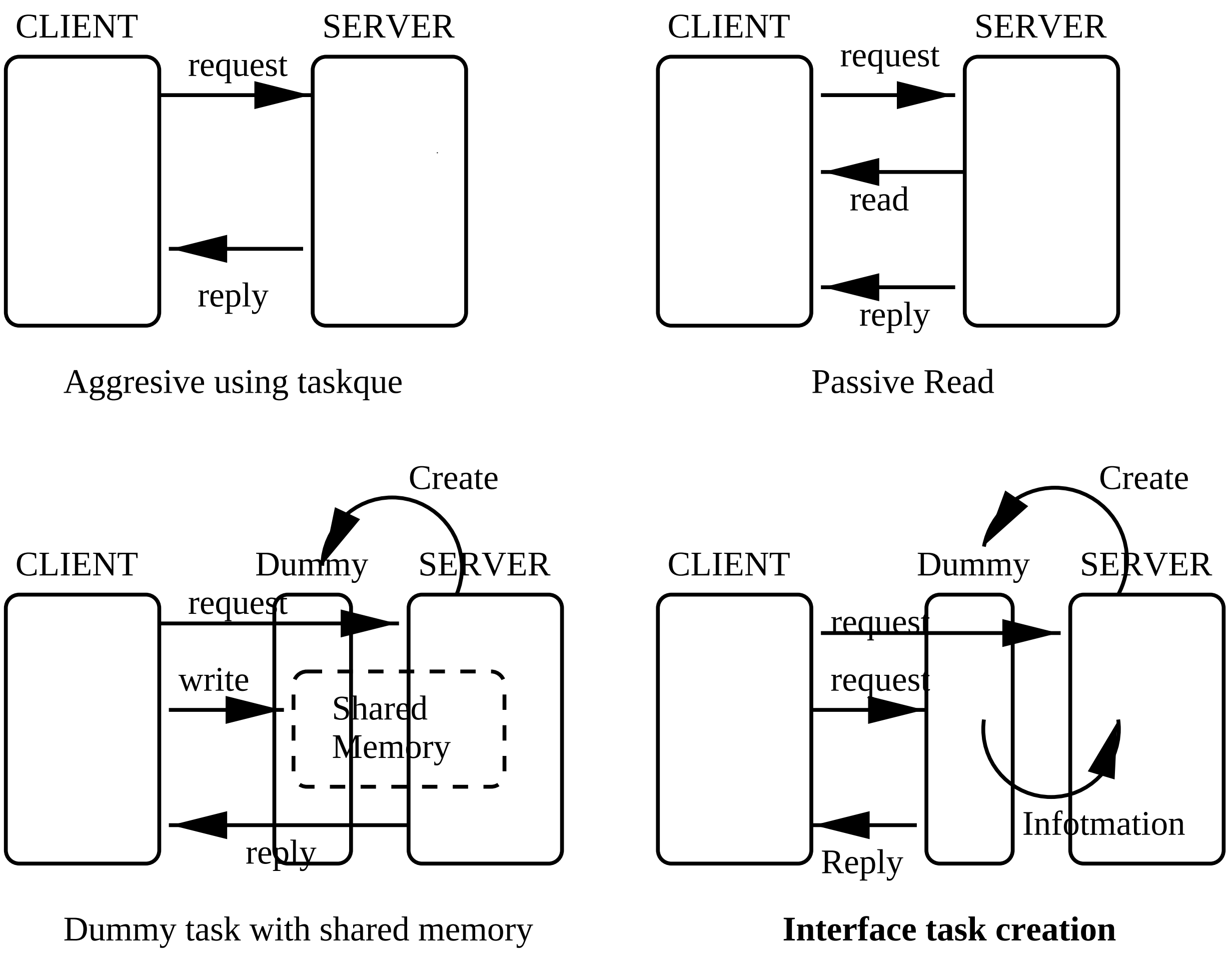

Figure 4.2: Four styles of Client-Server with the MBCF

- A server doesn't have to pass its access-key to its clients. This makes coding very simple.
- Processing of requests can be stated as soon as arrival of requests.

Disadvantage of this approach is

- Calling systemcall "read-taskque" and "send-taskque" is much slower than other light-weight systemcall of the MBCF.
- Connection establishment costs.(Creating Ltask table etc...)
- A data size of a request is restricted to a single packet size.

From these characteristic,this approach is good for the service without continuous requests and large request data size.Coding of this style is very similar to UDP coding style.

4.1.3 Passive READ

When client sends a connection request packet"WRITE_TASKQUE",information of its request is packed in the same packet.Typically,information contains memory address of request buffers of a client task.The server task reads requests of the client task from request buffers on the client's memory.

Advantage of this approach is

- A server doesn't have to pass its access-key to its clients.
- In comparison with "Aggressive use of taskque" style,size of data is not restricted and a sequence of requests are processed at one connection establishment.

Disadvantage of this approach is

- A server is very passive.A transaction for getting new request is processed by a server.This reduces performance of service.

With this approach,a server can process continuous requests at one transaction. When a server or client can estimate a sequence of continuous requests, this approach works very well.If not,with this approach,the cost of server's transaction to client for getting requests should be very large because costs for transaction are much larger than processing costs of requests in general.

4.1.4 Dummy task with shared memory

To make use of the power of the MBCF, the client and the server have to use "MBCF_WRITE"or MBFifo as a basic transaction method.To inform the client of an access-key is needed to use the MBCF for a transaction.But this is very dangerous because the client which knows the server's access-key can access an arbitrary memory area of the server task. A method for protecting a server is necessary to create safe client-server system.

To protect a server task, it is necessary to separate memory area for the MBCF transaction from the server task's memory space and separated memory area has different access-key and memory context. To create a separated memory,it is easy to make use of another task and shared memory. The server task create a new task (called dummy task) and create a shared memory area between itself and its dummy task.The dummy task essentially has a different memory context and a different access-key.

On SSS-CORE,it is one way to make use of systemcall "proc_fork()". This systemcall creates a new task which is a new task entity and has a new memory context.A new task's address space is a copy of its creator.The new task and the creator map a same new memory area to be shared. The new task's Ltask table is a copy of its creator's one. In this case,process of connection establishment is the following.

1. A client send a connection request to a server.
2. The server receive a packet from taskque and calls "proc_fork()" to create a dummy task.
3. The server and its dummy task map a new memory area as a shared memory area.And the dummy task informs the client of its name and access-key.
4. The client task send a connection establish request to the dummy task.

At this point, connections during the client task and the server task,the dummy task are established.The client send a request packet to a shared memory area of the dummy task making use of the MBCF.The server task keep a watch on shared memory area and get request.If necessary,the response of the request is returned by the dummy task. (To make use of explicit acknowledge function of the MBCF,the sender have to set its access-key.)

Another way is that a dummy task is created before a request from a client. A client knows the dummy task's name and send a connection request packet to the dummy task and the dummy task informs the client of its access-key.Of course,the dummy task maps a shared memory area among the server task.The server task read requests from a shared mapped area and response through the dummy task with inter-process communication and the MBCF. This method is simpler than using proc_fork(). In this method,because a dummy task communicates with a several clients,a dummy task has to reject read-requests of the MBCF.

In this method, a dummy task is an only memory-window and an agency of a transaction. All of services are executed by the server task itself.

4.1.5 interface task creation

This method is that a server creates a (child) task which offers a service for a client. We call this created task “interface task”.A interface task is a completely s separate task from server task and they communicate each other through inter-process communication,queue,message box, shared memory,MBCF etc,to say, data to be protected is stored in the server task and the interface task is an interface to the client task.

In this case,it is reasonable for the interface task to make use of MBSignal because MBSignal is advantageous on dispatching latency to other the MBCF primitives.A interface task registers necessary MBSignal procedures at first and A server task have no MBSignal service to the client. Process of connection establishment is the following.

1. A client task sends a connection request packet to a server task.
2. If necessary,the server task creates a interface task and it and informs the client of the interface task’s name.
3. The client task sends a connection request packet to the interface task.
4. The interface task informs the client of the information about its interfaces.

In view of the protection,it is important to reject read-requests of the MBCF or keep correspondence of one client to one interface task.

4.2 RPC/MBFifo

RPC/MBFifo is a RPC library which uses Memory Based Fifo as a transport layer. It is a transplantation of Sun RPC on the MBCF.Although Sun RPC provides users UDP or TCP as its transport layer,RPC/MBFifo provides transport layer of UDP, TCP or MBFifo.The implementation of client of RPC library is very similar to them of UDP or TCP but the server codes are very different because of protection mentioned in the former section.RPCGEN,protocol compiler for Sun RPC can be also used to create RPC codes automatically from protocol definition written by RPCL.

4.2.1 Difference between MBFifo and UDP,TCP protocol

In this section,we make it clear difference between MBFifo and UDP,TCP as a protocol for network transaction from a viewpoint of a programmer.

First,UDP is a connection-less protocol and doesn't guarantee an arrival of a packet UDP receive port can receive packets from any ports. A chunk of data user passed to an interface is directly transfered as one UDP packet. (datagram service) A too long packet is divided into packets of suitable size by IP layer. A target address of a UDP packet is a combination of an IP address and a port number.

Next,TCP is a connection oriented protocol and guarantees an arrival and an order of data. A client has to establish connection with a server before transmission. TCP is a protocol dedicated to one to one transaction ,that is, one port can connect only to one port simultaneously. TCP provides stream service and a user cannot and needn't recognize a packet. A target address of a TCP packet is a combination of an IP address and a port number.

Finally,MBFifo is a connection oriented protocol and guarantees an arrival and an order of packets. The MBFifo receive buffer can receive packets from any connected tasks. (Because the MBCF is essentially memory copying,a concept of "port" makes no sense on the MBCF.) A chunk of data user passed to an interface is directly transfered as one MBCF packet. A user has to be careful with limitation of packet size because the MBCF interface rejects too long data size request.(datagram service) A target address of a MBFifo is a combination of an logical task (a combination of a node ID and a task name) and a logical address of structure for memory based fifo. Table 4.1 summarize these differences.

Therefore,for programmers, (1)To use MBFifo,a task has to establish a connection with a target task before starting a transaction.It guarantees an arrival and an order of packets. On these two points, it is similar to TCP. (2)A user is aware of a packet with the MBFifo.And a receive port of the MBFifo can receive packets from several tasks.On these points,it is similar to UDP.

Protocol	Guarantee of arrival	keeping order	datagram service	one to one connection
UDP	No	No	Yes	No
TCP	Yes	Yes	No	Yes
MBFifo	Yes	Yes	Yes	No

Table 4.1: Comparison among UDP,TCP,MBFifo

4.2.2 Implementation of RPC/MBFifo

For the implementation of the RPC/MBFifo,we use variation of "dummy task with shared memory"method which is mentioned in the former section.

A server is consist of two tasks,a service task and an interface task. A service task and an interface task shares buffer for request MBFifo. The service task keeps watch on its request buffer and the interface task keeps watch on its taskque. The name of the interface task is registered in portmapper's service table. A "portmapper" program of Sun RPC informs a client of a port number of a service which a client wants to access, on the other hand, a portmapper of RPC/MBCF informs a client of a task name of a service.

A basic process of RPC/MBFifo transaction is the following.(See Figure4.3)

1. a client send a connection request to an interface task.
2. when the interface task receives connection request from the client, it informs the client of its access-key and address of its MBFifo buffer.
3. the client sends a request to the MBFifo buffer.
4. the service task read request from the MBFifo buffer and deal with it.
5. the service task send reply to the client.
6. A client send a connection-close request to the interface task at the end of request.

(Note:In the present implementation ,the service task creates a logical task entry for the client from physical node ID and the name of the client by a special systemcall "set_newtask()". In the future,"set_newtask(target_task,access-key)" will be substitute for current "set_newtask()". This systemcall sends an authentication packet to the target task.)

RPC/MBFifo has the following characteristics.

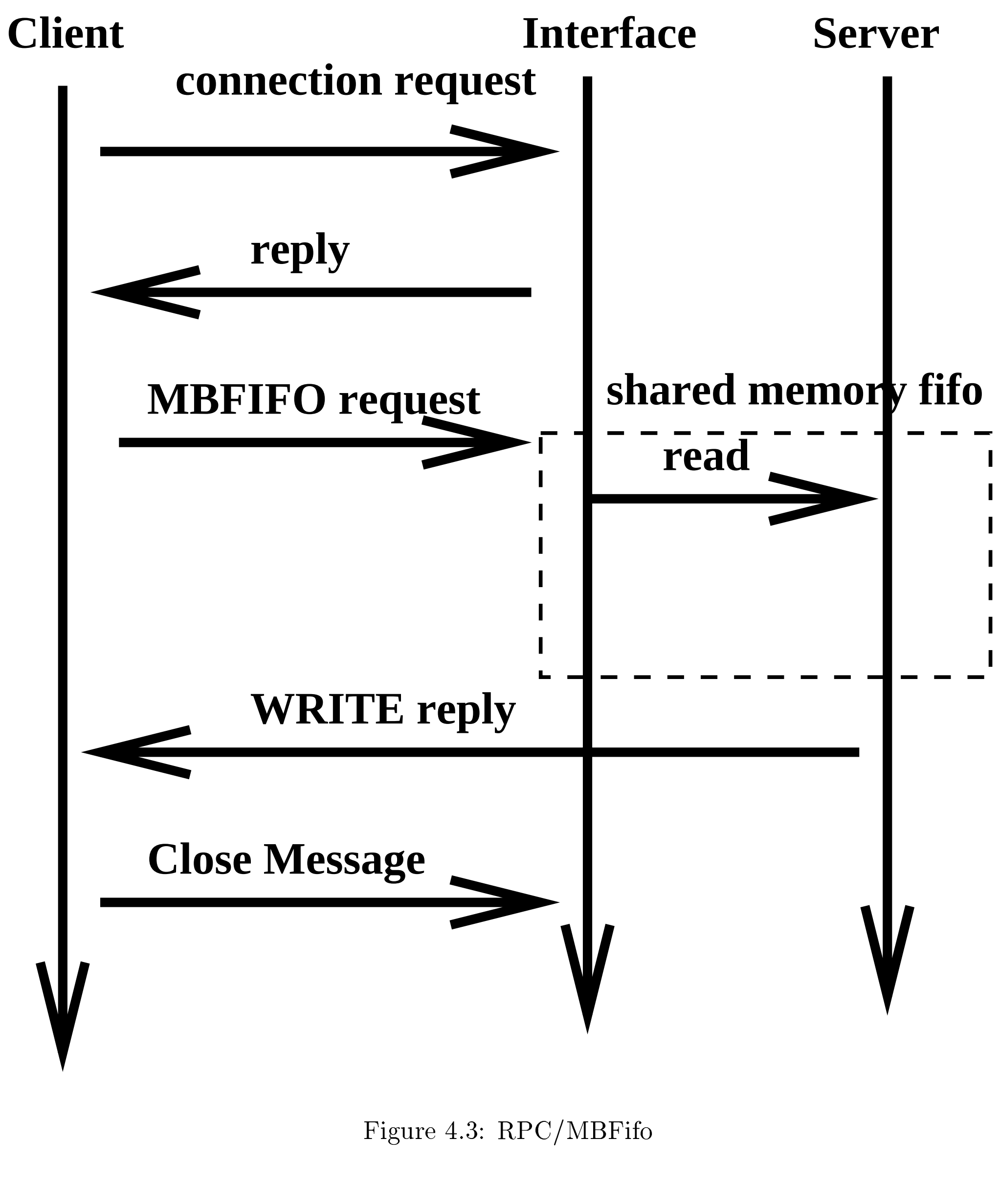

Figure 4.3: RPC/MBFifo

- Because of using the MBCF, RPC/MBFifo provides reliable execution.
- A request packet are delivered by MBFifo.And a reply packet is delivered by MBCF_WRITE directly to client's reply buffer which is specified in request packet.
- Because of reliability of transport layered,(the MBCF,a user of RPC/MBFifo can select "Blocking RPC" or "Non-Blocking RPC".
- For avoidance to overflow server's fifo buffer, RPC/MBFifo uses explicit-acknowledge function of the MBCF.
- RPC/MBFifo library routine has a common interface with Sun RPC,and RPCGEN of Sun RPC can be used to generate stub codes automatically.Marshaling/Unmarshaling routine is XDR.
- RPC/MBFifo library routine also provides UDP/TCP routine of Sun RPC. A server task can deal with UDP,TCP,or the MBFifo simultaneously.

4.3 RPC/MBSignal

RPC/MBSignal is a RPC library which is a interface to make use of the MBSignal as RPC.

The MBSignal provides the service of a remote invocation of a procedure. A user can invoke remote procedure registered by a server task by the MBSignal.

To use the MBSignal,a user has to specify a name of target task,access-key,and a memory address of the target procedure's memory-based-signal.

One purpose of RPC/MBSignal is standardization of a general client-server system using the MBSignal. A service procedure of MBSignal is asynchronously invoked whether a server is in sleep-state or not, a server application can be always asleep and has not to keep watch on request buffer.A server with MBSignal can use its scheduling time effectively.

A service lookup system is provided as a library routine as same as portmap service of Sun RPC.A user can search the service with three parameters ,PROGNUM,VERSNUM,PROCNUM.

All of request is transfered by MBSignal and all of result is transfered by MBCF_WRITE.This method helps executing Non-Blocking RPC.

4.3.1 Implementation of RPC/MBSignal

A server of RPC/MBSignal consists of two procedures,a server task and an interface task.We use "service task creation" method in the former section and while server tasks of RPC/MBFifo has a shared memory for MBFifo,servers of RPC/MBSignal don't have to create a shared memory.

Usually, all of requests from a client is processed by procedures of an interface task.If an interface task has to access an important data which is stored in a server task,it communicates with its server task by an inter-process communication mechanism,queue,the MBCF and so on. Process of RPC/MBSignal transaction is the following.(Figure4.4)

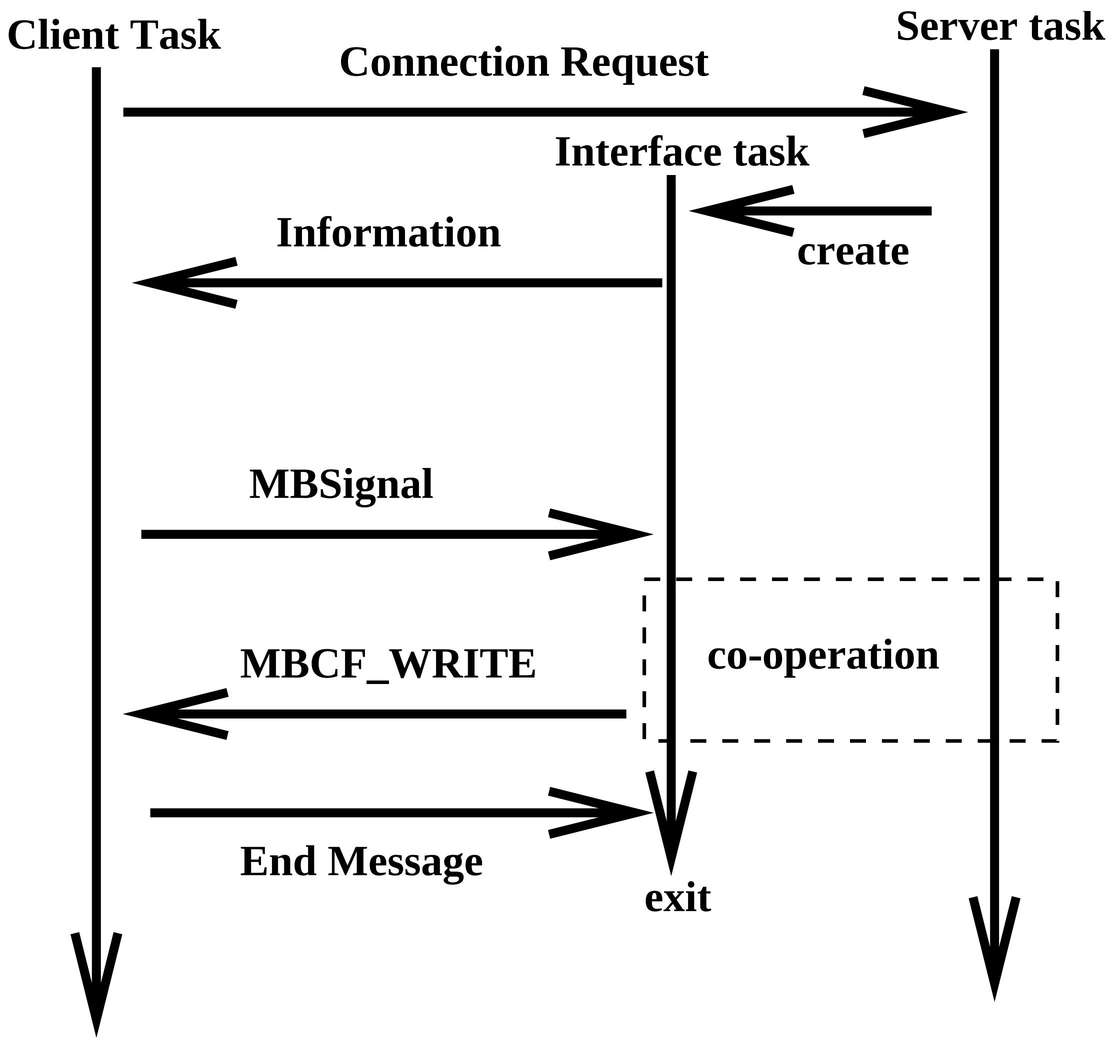

Figure 4.4: RPC/MBSignal

1. A server starts. It knows a name of an interface task's code.
2. A client send a connection request to the server task.
3. The server task creates an interface task.At this time, it informs an interface task of necessary informations.(passing the contents of request packet.)
4. The server task delete Ltask entry for the client task.
5. The interface task informs the client of a name and an access-key of itself and a memory address of MBSignal structure and a request buffer for requested service.
6. The client sends its request.Reply for the RPC request is directly written by an interface request with MBCF_WRITE.
7. At the end of requests,the client task sends a connection delete packet. And the interface task receives a connection delete packet and eliminates itself.

(Note: this implementation uses "set_newtask()",too.)

Because a server task does not give its access-key to the client task, if a server task doesn't use the MBCF for communicating with other task, ,a server task is completely protected from other tasks.

Because a procedure of the MBSignal can be invoked even if a target task is in sleep state, an interface can sleep forever after initialization.(No spin wait) All of scheduling time of an idle server is SS-WAIT on taskque.

Additionally,a server can select a several interface task code considering who a client task is.

The RPC library provides the following routines.

RPC client routine Client RPC library has a same interface to the Sun RPC.

Portmapper Portmapper server and client routine is available.

An interface task routine. An initialization routine and RPC dispatcher,RPC register routine ..etc is available. Connection management procedure for MBSignal is automatically registered.

A server task An interface task creation routine is served.

A protocol compiler like RPCGEN is not prepared today.

4.3.2 Characteristics of RPC/MBSignal

RPC/MBSignal has the following characteristics.

- Because of using the MBCF as a transport layer, it provides a reliable execution. RPC/MBSignal supports both "Blocking" and "Non Blocking" RPC.
- A RPC request packet is delivered by MBSignal and a reply packet is delivered by MBCF_WRITE directly to the reply buffer.
- All of transaction data is marshaled by user's structure. Whether a user uses marshaling code or not is his business.
- A structure of RPC/MBFifo library routine is different from that of the Sun RPC. A programmer of client routines for RPC/MBSignal can use same manner when he programs with Sun RPC.But A programmer of server routines has to write codes for an interface task and a server task.Most of necessary routines are provided as library routine and RPC registration routine etc... has a same interface with the the Sun RPC server code.
- MBSignal execution frequency parameter is "FIRST_ONLY_NONBLK' and all of arguments of MBSignal is passed by a fifo interface
- Buffer for request and Buffer for argument is separated.A client sends a MBSignal request and arguments to the separated address.A client task send an argument packet first and request packet next.This policy reduce unpacking cost of request.
- All of registered procedures of RPC is dispatched by a dispatching procedure on MBSignal according to requested three parameters of Sun RPC, PROGNUM,VERSNUM and PROCNUM.

Chapter 5

Performance evaluation of RPC/MBCF

5.0.3 Round-trip latency of RPC/MBCF

5.0.3.1 Environment for measurement

Performance of RPC/MBCF is measured in the following environment.

- Axil 320 model 8.1.1 (Sun SparcStation 20 compatible,85MHz,SuperSPARC $\times$ 1)
- Sun Microsystems Fast Ethernet SBus Adapter 2.0
- Bay Networks BayStack 350T half-duplex mode **100Base-TX**
- a scalable operating system : SSS-CORE ver1.0

5.0.3.2 Sample program for measurement

We used sample program "remote_strlen" for this measurement. A client sends a string to a server, a server culculates the length of it and reply its length to the client. A service procedure for RPC/MBFifo is

```
transfer_res *transfer_1(datatype *data)  /* datatype = char */
{
  static transfer_res result;
  result.errno = 0;
  result.transfer_res_u.length = 4;
  return &result;
}
```

This code has a same manner with service procedure for Sun RPC. Caluclating length of string doesn't executed in this procedure.

A service procedure for RPC/MBSignal is

```
void str_return(data,di)
  char *data;
  DISPATCH_INFO *di;                /* information of request */
{
  int length = 4;
  br_reply(di,(char *)&length,4);   /* Send reply */
  return;
}
```

RPC/MBSignal doesn't uses XDR routine.

Cost of calling XDR routine for this procedure is memory copying and caluclaton of length of string at client side. RPC/MBFifo's client routine caluculates length of string at each request. Because of XDR,the number of times RPC/MBFifo execute copying memory is essentially greater than that of RPC/MBSignal by two(one at client encoding,one at server decoding).XDR routines are possibly inlined to eliminate fuction calls.

A server of RPC/MBFifo does SS-WAIT with keeping watch on its MBFifo. If there is no request,a server go to sleep for one tick(10msec).

Code for RPC/MBFifo uses automatically generated stub code from RPCGEN. Code for RPC/MBSignal was hand coded but aggressively uses library routines.

We also measured Round-trip latency of Sun RPC using SunOS instead of SSS-CORE. Code for Sun RPC uses automaically generated stub code from RPCGEN.This code is same to MBFifo's one except for systemcalls are SunOS's systemcall and using UDP/TCP.

5.0.3.3 Round-trip latancy

Measured minimum round-trip latency of sample codes are shown in Table 5.1 and Figure 5.1.Data size does not include length of RPC header.

Compareing RPC/UDP,RPC/TCP, and RPC/MBFifo, difference between the performance of the MBCF and that of UDP/TCP appears to the difference of RPC. On RPC/UDP/TCP influence

Figure 5.1: Minmum Round-Trip Latency of RPC/MBCF

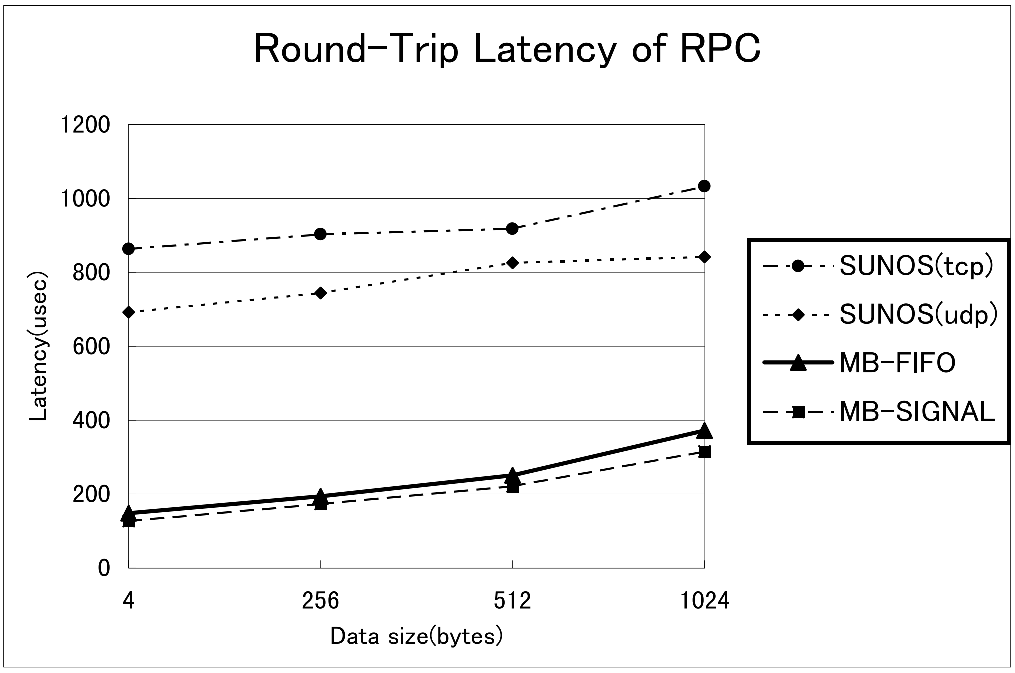

Table 5.1: Minimum Round Trip Latency of RPC/MBCF

data size (byte)	4	256	512	1024
RT of RPC/MBFifo (μs)	148	194	251	372
RT of RPC/MBSignal (μs)	127	173	221	315
RT of RPC/UDP(SunOS)(μs)	605	661	719	797
RT of RPC/TCP(SunOS)(μs)	712	790	794	850

of data size appears smaller than that of RPC/MBCF. This fact implies how light the MBCF's interface routines are.

See RPC/UDP and RPC/MBFifo,RPC/MBSignal.Although UDP does not provide reliable service, and has no packet-stream management routine,the MBCF is much faster than UDP. In other words,RPC/UDP has doesn't provides reliable routine and RPC/MBCF provides reliable services,but RPC/MBCF can provides faster service than RPC/UDP.

Round-trip latency of the MBCF_WRITE for 64byte data is measured as 60 μs. (Header length of RPC/MBSignal request is 40 bytes,and) (See Table 3.2)This latency shows latency between calling the MBCF and receiving explicit ack of the MBCF. This ack is automatically returned from target nodes' interrupt routine. The latency of RPC/MBSignal shows the latency of the following process.

1. [Client]Call RPC/MBSignal routine.
2. [Client]Send a request packet and an argument packet to the target.(these two are combined and into one packet)
3. [Server]MBSignal routine wakes up and pull out request from its request fifo.
4. [Server]Search and Dispatch a service routine
5. [Server]Send a Reply packet
6. [Client]Receive reply. This phase uses SS-WAIT.

Considering conbining of packets,total numbers of packets is twice as many as that of experiment of MBCF_WRITE.The handler routine of MBSignal is enough light.MBFifo has a disadvantage of SS-Wait's sleeping and scheduling.

(Note: Measurement environment of round-trip latency is not completely same on the two experiment.)

Chapter 6

Concluding Remarks

We discussed the implementation of RPC on the MBCF.

The key point for server protection on the MBCF is not to give an access-key and separation of interface memory area and server memory area. If a server deal with plural clients simultaneously, it is important to protect each client's request from another task. A server should not give a read-access permission to its clients.

There is much benefit to use the MBCF as a transport layer of RPC. First,because the MBCF provides low round-trip transaction, RPC/MBCF's round-trip latency is naturally low. Next,because the MBCF guarantees an arrival of packets and keeps an order of packet on a connection,RPC/MBCF can provide a reliable service,"one result to one request"semantics RPC. Because the MBCF has no restriction of "port", an implementation of "Non-Blocking RPC" was very easy,to say, classification of returned results and copying it to user's buffer at the client's receiving port is not necessary at all. In addition,because the MBCF is copying method from one's logical address to another's logical address,a server program of the RPC/MBCF can make direct use of a pointer which indicates a memory address of a client.

We introduced RPC/MBFifo and RPC/MBSignal as RPC/MBCF.

RPC/MBFifo is a re-implementation of Sun RPC using the MBFifo as its transport. It provides the same function as Sun RPC which provides services on UDP or TCP. By the benefit from the MBFifo and the Sun RPC, RPC/MBFifo provides reliable,high-performance,easy to use RPC. A protocol compiler RPCGEN and a several library routines are available. Addition to functions of Sun RPC,it also provides "Non-Blocking RPC" service and reliable "one reply per one request" semantics.

RPC/MBSignal is a library routine for using the MBSignal in the style of generally used RPC. RPC/MBSignal provides a procedure searching system and protocol for request/response on MBSignal. It also provides “Non-Blocking RPC” service and reliable “one reply per one request” semantics,too.

The round-trip latency of RPC/MBFifo is twice as fast as that of Sun RPC on UDP/TCP, and the round-trip latency of MBSignal is lower than that of RPC/MBFifo.

RPC/MBCF provides a reliable and high-performance RPC compared with RPCs which is implemented on UDP or TCP transport layer. It also provides “Non-Blocking” RPC and reasonable execution semantics of RPC.

Consequently, experiment and performance evaluation of RPC/MBCF in this thesis stands for that the MBCF provides high-performance computing on client-server applications.

References

[1] A. L. Ananda, B. H. Tay, and E. K. Koh. A Survey of Asynchronous Remote Procedure Calls. *sigops*, 26(2), April 1992.

[2] Andrew Birrell and Bruce Nelson. Implementing remote procedure calls. *ACM Trans. Computer Systems*, 2(1):39–59, February 1984.

[3] John Bloomer. *Power programming with RPC.* O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, February 1992.

[4] Peter F. Corbett and Dror G.Feitelson. The vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, August 1996.

[5] John H. Hartman and John K. Ousterhout. The zebra stripped network file system. *ACM Transactions on Computer Systems*, 13(3):274–310, August 1995.

[6] Sun Microsystems Inc. RFC1094: Network filesystem specification, March 1989. Status: PROPOSED STANDARD.

[7] Sun Microsystems Inc. RFC1813: NFS version3 protocol specification, March 1989. Status: PROPOSED STANDARD.

[8] Orran Krieger and Michael Stumm. Hfs: A performance-oriented flexible file system based on building-block compositions. *ACM Transactions on Computer Systems*, 15(3):286–321, August 1997.

[9] J. Niwa T. Inagaki T. Matsumoto and K. Hiraki. Supporting software distributed shared memory with optimizing compiler. In *PDPTA'98*, pages 225–234, August 1998.

[10] Kenji Morimoto Takashi Matsumoto and Kei Hiraki. Implementing mpi with the memory-based communication facilities on the sss–core operating system. In *Proc. of 5th European PVM/MPI Users' Group Meeting*, pages 223–230, September 1998.

[11] Takashi Matsumoto and Kei Hiraki. Mbcf:a protected and virtualized high-speed user-level memory-based communication facility. In *Proceedings of ICS*, pages 259–266, july 1998.

[12] A.D.Birrell R. Levin R.M.Needham and M.D.Schroeder. Grapevine:an exercise in distributed computing. *Communications of ACM*, 25(4):260–274, April 1982.

[13] Thomas E. Anderson Michael D. Dahlin Jeanna M. Neefe Daid A. Patterson Drew S. Roselli and Randolph Y. Wang. Serverless network file systems. *ACM Transactions on Computer Systems*, 14(1):41–79, February 1996.

[14] Michael D. Schroeder and Michael Burrows. Performance of firefly rpc. *ACM Transactions on Computer Systems*, 8(1):1–17, February 1990.

[15] R. Srinivasan. RFC 1831: RPC: Remote procedure call protocol specification version 2, August 1995. Status: PROPOSED STANDARD.

[16] Hal Stern. *NFS & NIS*. O'Reilly & Associates, Inc., 1991.

[17] Sun Microsystems, Inc. RFC 1050: RPC: Remote procedure call protocol specification, April 1988.

[18] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, 1992.

[19] B. H. Tay and A. L. Ananda. A Survey of Remote Procedure Calls. *ACM Operating Systems Review*, 24(3):68–79, July 1990.

[20] P. Weiss. *Yellow Pages Protocol Specification*. Sun Microsystems, Inc., 1985.

[21] 松本 尚 and 平木 敬. 100BASE-TX によるメモリベース通信の性能評価. In コンピュータシステムシンポジウム論文集, pages 101–108. 情報処理学会, November 1997.

[22] 松本 尚, 駒嵐 丈人, 渦原 茂, 竹岡 尚三, and 平木 敬. 汎用超並列オペレーティングシステム： SSS-CORE — ワークステーションクラスタにおける実現 —. In 情報処理学会研究報告 *96-OS-73, Vol. 96, No. 79*, pages 115–120. 情報処理学会, August 1996.

[23] 松本尚, 古荘進一, and 平木敬. 汎用超並列オペレーティングシステム SSS–CORE. 日本ソフトウェア科学会第 *11* 回大会論文集, pages 13–16, Oct 1994.