ISIPC: Instant Synchronous Interprocess Communication

Toshihiro Yamauchi\textsuperscript{1}, Kazuhiro Fukutomi\textsuperscript{2,}, Hideo Taniguchi\textsuperscript{1}

\textsuperscript{1}Graduate School of Natural Science and Technology, Okayama University
3-1-1 Tsushima-naka, Kita-ku, Okayama 700-8530, Japan
{yamauchi, tani}@cs.okayama-u.ac.jp

\textsuperscript{2}Graduate School of Information Science and Electrical Engineering, Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan

\textsuperscript{*}Presently, the author is with Computer Architecture \& Security Systems Laboratory
Corporate Research \& Development Center, Toshiba Corporation.
doi:10.4156/jnit.vol1.issue3.9

Abstract

Interprocess communication (IPC) is often used to exchange data between cooperative processes, and the performance of IPC largely determines the processing time of application programs. Moreover, it is used for most of the kernel calls in a microkernel-based operating system (OS). Therefore, the performance of IPC affects the performance of the OS. In addition, the completion of the message-passing mechanism has to be indicated by executing a receive operation in order to maintain synchronization. Thus, two operations are required in this mechanism to complete the communication. On the other hand, no receive operation is required to indicate the completion of the communication in the case of asynchronous communication; however, in this case, no proof of the data being received is provided to the sender process.

In this paper, we propose an instant synchronous interprocess communication (ISIPC) mechanism that can achieve both instantaneous communication and data synchronization. ISIPC has two functions: push function and sack function. We describe the design of the ISIPC mechanism and also its implementation on the Tender operating system. In addition, we present the evaluation results for the ISIPC mechanism.

Keywords: Interprocess Communication, Synchronization, Operating system

1. Introduction

The performance of computer hardware has been continuously improved. However, service software packages have become increasingly complicated and involve more than one process. Thus, in order to improve service performance, efficient interprocess communication (IPC) is required. IPC is a technique for the exchange of data between two or more processes of an operating system (OS). It is frequently used in cooperative processing among application programs (APs). It also influences the performance of an AP.

Moreover, it is used for most of the kernel calls in a microkernel-based OS [1]. Therefore, the performance of IPC affects the performance of the operating system. In one study [2], an adaptable and extensible OS based on the L4 microkernel [3] and multi-server scheme was developed for control-system software. In other OSes based on the microkernel [4], [5], [6], since service of the OS is implemented by using a user-level process, the overhead of the IPC should be reduced.

The message-passing mechanism is widely used for communicating data. The completion of the message-passing mechanism has to be indicated by executing a receive operation in order to maintain synchronization. Thus, two operations are required to complete communication by this mechanism. On the other hand, no receive operation is required to indicate the completion of the communication in the case of asynchronous communication; however, in this case, no proof of the data being received is provided to the sender process.

In this study, we propose an instant synchronous interprocess communication (ISIPC) mechanism that can achieve both instantaneous communication and data synchronization. The ISIPC mechanism involves two functions: push function and sack function. The push function facilitates the transmission of communication data to a receiver process without requiring a receive operation. In addition, the
sender process can be simultaneously synchronized with the receiver process. The sack function facilitates the sending of a request by a receiver process for the sack operation. As a result of this sack operation, the sender process transmits communication data to the receiver process, and the receiver process can receive the communication data irrespective of the state of the sender process.

In this paper, we describe the design of the ISIPC mechanism. We also describe the implementation of the proposed ISIPC mechanism on The ENduring operating system for Distributed EnviRonment (Tender) OS [7], [8]. In the case of Tender, objects to be controlled and managed by the OS are known as “resources,” and they have resource identifiers and resource names. We deploy the ISIPC mechanism involving the push function and sack function. We describe the evaluation results for the ISIPC mechanism. The results show that the sack function is suitable for the imprecise computation model [9].

2. Instant synchronous interprocess communication

The ISIPC mechanism involves two functions: push and sack. Figure 1 shows the operations of these two functions. In the case of the push function, communication data in the virtual address space of the sender process are moved to the virtual address space of a receiver process. In addition, the sender process can simultaneously synchronize the push function with the operation of the receiver process. To facilitate this synchronization, the receiver process can initiate an operation when the push function is invoked. In the case of the sack function, the communication data in the virtual address space of the sender process are moved to the virtual address space of the receiver process.

The following are the advantages of the ISIPC mechanism:

(1) ISIPC is implemented in the OS layer. Thus, the OS provides an atomic operation for IPC.
(2) The OS can instantly initiate an operation when it receives an ISIPC request. In addition, the OS can ensure that the operation is completed without any interruptions by other processes.

![Figure 1. Push function and sack function](image)

This mechanism can facilitate both instantaneous communication and instantaneous synchronization as follows:

(1) When a sender process (push process) invokes the push function, the sender process can synchronize with a receiver process. The receiver process does not confirm reception. In addition, the receiver process can start the processing of the data received while synchronizing with the sender process.

(2) The receiver process (sack process) invokes the sack function to synchronize with the sender process. This operation does not include a wait time, and thus instantaneous synchronization can be achieved.

(3) It is possible to clarify the imprecise computation model by using ISIPC. Either a sender process or a receiver process initiates the communication. The other process does not need to wait for synchronization.

3. Implementation

ISIPC is implemented on Tender using the “container” and “event” resources.
3.1. Tender operating system

The objects to be controlled and managed by Tender are known as “resources.” Resources have identifiers and names for operations. The resource identifiers and resource names include location information that indicates the particular machine on which the OS is installed. Figure 2 shows the structure of a resource identifier and resource name. Resource names have a tree structure. An example of a resource name is “/machine1/process/procA.”

![Figure 2. Resource identifier and resource name](image)

The interface for the operation of resources is a unified interface named the Resource Interface Controller (RIC). Programs that use resources are called through the RIC. The programs that use each resource are separated from each other. In addition, the sharing of a program component by multiple programs that use resources is not allowed. The program modules consist of five program components “open,” “close,” “read,” “write,” and “control.” The RIC has a pointer table that includes all the pointers of the program components. Each program that manages resource has to call any program component through the RIC. Bypassing the RIC is prohibited in the Tender kernel.

There is a separate management table for each resource. In addition, pointers referring to values within the other resource management table are prohibited. The existence of an individual resource does not depend on the existence of other resources or processes because the management table for each resource is separate. In other words, the resources are independent of each other.

Since the individual resources are independent of each other, Tender is able to recycle them. Thus, Tender can preserve process resources instead of deleting them at process termination, and it can recycle the preserved process resources during process creation. Therefore, by preserving and recycling process resources, users can realize a reduction in the cost of process creation and termination [10].

3.2. Functions of ISIPC

Table 1 lists the functions related to ISIPC. There are two functions for the push and sack functions. The function ctainerpush is the push function, and ctainersack is the sack function. The functions eventsend and eventattach are used in ctainerpush for the synchronization of the sender process with the receiver process.

The arguments of the functions “ctainerid,” “vmid,” and “envtid” indicate the resource identifiers of the container resource, virtual space resource, and event resource, respectively. The argument “reqaddr” indicates the virtual address space of the receiver process; in ctainerpush, the sender process requests that the container resource be pasted at this virtual address space. The argument “reqaddr” indicates the virtual address space of the receiver process; in ctainersack, the receiver process requests that a container resource be pasted at this virtual address space. The modes of these functions are
“move,” “copy,” and “share.” These functions use the “event” resource to facilitate the synchronization of the sender and receiver processes with each other.

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>containerpush(containerid, op, reqaddr, vmid, evntid, *paddr)</td>
<td>This function pushes container(containerid) with mode(op) to a receiver process. Then, it sends event(evntid) to it.</td>
</tr>
<tr>
<td>containersack(containerid, op, reqaddr, *paddr, *size)</td>
<td>This function sacks container(containerid) with mode(op) from a sender process.</td>
</tr>
<tr>
<td>eventsend(evntid)</td>
<td>This function sends an event(evntid). If the event is attached to a function, the function is executed.</td>
</tr>
<tr>
<td>eventattach(evntid, func)</td>
<td>This function attaches an event(evntid) to a function(func).</td>
</tr>
</tbody>
</table>

### 3.3. Interprocess communication on Tender

Figure 3 shows the functioning of IPC on Tender. The container resource for IPC is present in the virtual address space.

In the case of the push function, process A pushes a container resource to process B. Then, it invokes a function by using the event resource. The function is executed on the virtual address space of process B.

In the case of the sack function, process C sacks a container resource from process B. The container resource is sent to the virtual address space of process C.

In the send-and-receive function, a container resource is sent to a container box by process C. Then, the receiver process D receives the container resource from the container box. The container box has a FIFO queue and manages container resources in FIFO manner. When a container box receives a container resource, the container is inserted in the FIFO queue. When the container box obtains a request for a receive operation, a container resource is removed from the FIFO queue.

AP developers consider the following points:

(1) Address collision

Address collision may occur in ISIPC operations, which will cause errors in the operation. Therefore, a program must retry with a different address or attempt to solve the address collision problem.

(2) Data consistency

In the sack operation, a sack process may receive incomplete data. Thus, the AP should verify the integrity of the data.

![Diagram](https://via.placeholder.com/150)

**Figure 3.** Interprocess communication on Tender

### 3.4. Access contr
In order to protect the virtual address space of each process, an access control mechanism is also implemented. The mechanism controls and grants the ISIPC requests between user processes or between group processes.

4. Evaluation

4.1. Purpose of evaluation

To show the practicality of the proposed method, we evaluated the ISIPC mechanism on the Tender OS. We implemented the ISIPC mechanism in Tender and then performed experiments. The experiments were performed using a PC (CPU: Pentium 4, 2.0 GHz; OS: Tender).

In the basic performance evaluations, we measured the basic performance of the push function. The push function has three modes for communication. Each mode was evaluated in these experiments. Further, the performance of the share mode in the case of each communication method was evaluated. For evaluating the sack function, a program based on the imprecise computation model was used to verify the practicality of the sack function.

4.2. Basic performance evaluation

Figure 4 shows processing time of the container push function. In this experiment, container resource size was varied from 0 to 64 KB. Since the copy mode involves performing memory copy from a sender process to a receiver process, the processing time increases in proportion to the size of the container resource. On the other hand, because the move mode and the share mode do not involve performing memory copy, these modes are faster than the copy mode.

The performance of the copy and share modes can be analyzed from Figure 5, which shows the processing time of these two modes. As seen in the figure, the share mode is faster than the move mode. In the share mode, a container resource is attached to a receiver’s virtual address space. On the other hand, in the move mode, a container resource is removed from the virtual memory space of the sender process and then attached to the virtual memory space of the receiver process. Since only the attach operation is required in the case of the share mode, it is faster than the move mode.
ISIPC: Instant Synchronous Interprocess Communication
Toshihiro Yamauchi, Kazuhiro Fukutomi, Hideo Taniguchi

In order to verify the effectiveness of ISIPC, the push function and sack function are compared with the send-and-receive function by evaluating their share modes. Figure 6 shows the processing time of the share mode for each communication method. In the case of the send-and-receive function, the processing time is the total processing time of the send function and the receive function.

The push function and the sack function are faster than the send-and-receive function, because the send-and-receive function involves a send system call and a receive system call. On the other hand, the push function and the sack function each involve one system call. Therefore, the overhead for the push function and that for the sack function are less than that for the send-and-receive function.

The sack function is faster than the push function. In the push function, after a container resource is attached to the virtual address space of the receiver process, an event is sent to the receiver process. This increases the overhead for the push function. The sack function can receive a container resource without sending any event.

Figure 5. Processing time of container push function (move and share modes)

4.3. Evaluation of sack function

We evaluated the sack function by using a program based on the imprecise computation model [5] that calculates the approximate solutions of natural logarithms. We compared the sack function with the send-and-receive function. Figure 7 shows the flowchart of the program that uses the sack function. Figure 8 shows the flowchart of the program that uses the send-and-receive function.

Figure 6. Processing time of share mode for each communication method

4.3. Evaluation of sack function

We evaluated the sack function by using a program based on the imprecise computation model [5] that calculates the approximate solutions of natural logarithms. We compared the sack function with the send-and-receive function. Figure 7 shows the flowchart of the program that uses the sack function. Figure 8 shows the flowchart of the program that uses the send-and-receive function.
There are two processes in these programs. One process calculates the approximate solution. The other process receives the approximate solution by using the sack function.

In the sack function, the receive process can obtain the calculation result when it calls the containersack function. Further, the calculation program performs only calculation steps. In contrast, in the send-and-receive function, the receiver process has to wait until the result is sent by the calculation process. In addition, the calculation program must repeatedly check whether the result should be sent.

Table 2 shows the elapsed time and the number of calculation steps in the evaluation.

In the sack function, the calculation program does not need to check whether the result should be sent. However, in the send-and-receive function, the calculation program has to check the time at every calculation step. As observed, the processing of the program that uses the sack function is faster than that of the program that uses the send-and-receive function. Table 2 shows the difference between the number of calculated steps. The processing time for one step is approximately 12 ms. When the elapsed time is 300 s, the difference between the number of calculation steps for both the programs is 46 steps. For performing the calculation of these 46 steps, approximately 0.552 s is required.

The difference in the calculation steps increased with the elapsed time. Thus, the sack function is appropriate for a program that requires more time for calculation.
Table 2. Elapsed time and the number of calculation steps in the evaluation

<table>
<thead>
<tr>
<th>Elapsed time (s)</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>100</th>
<th>200</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation steps in the case of sack function</td>
<td>833</td>
<td>1669</td>
<td>3349</td>
<td>8474</td>
<td>17346</td>
<td>26727</td>
</tr>
<tr>
<td>Calculation steps in the case of send-and-receive function</td>
<td>833</td>
<td>1667</td>
<td>3345</td>
<td>8460</td>
<td>17319</td>
<td>26681</td>
</tr>
<tr>
<td>Difference</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>14</td>
<td>27</td>
<td>46</td>
</tr>
</tbody>
</table>

5. Related work

In a study on IPC that took priority and real-time processing into consideration. In paper [11], Mach IPC for real-time applications was extended such that priority inversion could be avoided. The evaluation results show that real-time IPC can prevent the priority inversion and improve CPU utilization for real-time applications.

In another study [12], a general transformation mechanism was proposed; the mechanism takes advantage of the temporal characteristics of the system to reduce both the time and space overheads of current single-writer multiple-reader algorithms. A reduction in the execution time by 17-66% and in the memory space by 14-70% was achieved when three wait-free algorithms were improved by applying the transformation.

6. Conclusion

Two functions, push and sack functions, have been proposed for realizing ISIPC. These functions correspond to IPC mechanisms and can be used to achieve both instantaneous synchronous communication and data synchronization. The OS can instantly initiate an operation when it receives a request for instantaneous IPC. In addition, the OS can ensure that the operation is completed without any interruptions by other processes.

We evaluated the effectiveness of ISIPC, the results of the evaluation show that since the move mode and the share mode do not involve performing memory copy, these modes are faster than the copy mode. The results also show that the push and sack functions are faster than the send-and-receive function, because the send-and-receive function includes a send system call and a receive system call, while the push and sack functions each involve a single call.

In addition, we evaluated the sack function by using the imprecise model program. It is possible to clarify the structure of the imprecise model by using ISIPC. The results showed that the sack function is faster than the send-and-receive function. When the elapsed time is 300 s, the difference is 46 steps. For performing the calculation of these 46 steps, approximately 0.552 s is required.

7. Acknowledgement

We would like to thank Hironori Namba of Okayama University for his support.

8. References


