Indian Journal of Science and Technology, *Vol 7(6), 839–847, June 2014*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

A New Approach for Inter Process Communication with Hybrid of Message Passing Mechanism and Event based Software Architecture

Farhad Soleimanian Gharehchopogh*, Esmail Amini and Isa Maleki

Computer Engineering Department, Science and Research Branch, Islamic Azad University, West Azerbaijan, Iran;
bonab.farhad@gmail.com, es.amini56@gmail.com, maleki.misa@gmail.com

Abstract

In this paper, we use software architecture style based on event-driven and message passing communication method and determine a framework for interaction among practicable processes on Operating System (OS). In the proposed method, the required data are sent to the process or other processes in a standard message frame and with determined structure to the OS, then, the OS distributes the received message considering its recipient processes in the system, rather a process communicates directly with other specific process. The major features of the proposed method include the synchronization among the processes, the simplicity of implementation, easy extensibility, remote access which finally can improve the interaction between the OS and processes. Along with producing the systems based on an integrated frame, we obtain a determined standard in Inter Process Communication (IPC) by mediating an OS.

Keywords: Event-driven, IPC, Message Passing, OS, Software Architecture

1. Introduction

The software architecture style based on event is one of the most successful and most used available architecture in designing extensible systems. In this architecture style, summoning the operation is separated from its operation as the applier of a service is independent from the provider and mainly doesn't know about it[1]. Even it is possible that in a distributed system, these two elements are operating in two separated processor. We use communication processes as the sub-group of independent component. In this architecture style, the general principles and rules of implicit innovation Event-Based systems are almost dominated. In this style, each element has a series of operations and events. The elements acted in a way to assign some of their elements to some of the events related to the system other elements in order to do operation as an event occurred[2]. Communication processes style is based on implicit summoning which means that a software element creates an event rather it summons a system directly. Then, it generally distributes the event in the total system[2]. So, by producing an event implicitly, a software elements cause to operate some operations. It is as the element can't determine which processes may be operated. In this architectural style, the main emphasis is based on message passing among software elements. This feature causes that concepts such as event occurrence and event general distribution take a specific meaning[3]. Event occurrence means the message delivery to a software system. For general distribution of its occurrence, it can be distributed by a message communication protocol among other elements.

In this paper, the processes are implemented in a way that firstly it arranges a message containing the necessary data to communicate with the other processes in a message frame based on XML structure and then sends

*Author for correspondence

it to available Event Bus in the OS core instead of a process wants to make relation directly with the other. Then, based on determined security decisions, the OS generally distributes the received message in the system and among the processes. The available Event Bus in the OS is acted based on event processing as the messages which exchange between OS and processes have a standard and defined structure in total system. As soon as the message enters to the Event Bus, the OS stimulates an event similar to the received message and generally distributes it. Due to the above mentioned structure for the OS, the processes must be in a way to adjust with the above structure and make relation with it. To do so, we design a standard connector for the system processes as when a program is changed to the process and its specific data recorded in the available processes tables in the OS, the OS provides the mentioned connector for each process. It includes an event processor and necessary buffers for sent and received messages to the process. The connector provides the ability of making relationship with the external environment without its affection on the internal structure.

In the proposed method, it is used the structure based on XML which involve high flexibility and processing rate. In message passing procedure among them, a process may reply to the received message. Consequently, it regulates the reply in a standard structure and almost similar to the sent messages structure and then sends it to the determined module. It causes more integrity between the processes and request and responses structure. The main features of the proposed method include the simplicity of implementation, easy expandability, remote access and the lack of necessity of synchronization among processes. By observing and improving detailed points in the proposed method, it can be reached to an acceptable efficiency in making relationship between the processes and the OS[3,4].

We organize the general structure of this paper as follow: In section 2, it is reviewed IPC mechanisms. In section 3, it is reviewed previous strategies about IPC. In section 4, we review the proposed strategies of IPC and its general structure. We also point out to its internal and external structure and review the internal modules, separately. In section 5, we introduce the proposed method and finally in section 6, it is introduced the conclusion and also future works and studies about IPC and specific the proposed method. We also apply Enterprise Architect7.0 to model the proposed method and provide UML charts.

2. Interprocess Communication (IPC)

In the OS, there are a lot of issues and processes which related to each other. In relation viewpoint, the available processes in the OS are classified in two groups of independent processes and cooperator processes. The processes, in which the following proportions are valid about it, will be an independent process[3,5]:

- Its beginning and ending will be possible every time and without affection on the other processes.
- The output of the process will be specified.
- If the process input is similar in different operations, the output will be, too.
- There is no available shared status.

If two processes don't have one of the above mentioned proportions, it is concluded that these two processes are in cooperation and need of each other data. As the processes didn't have access to the data space and their addresses due to some security reasons, so, they cooperated and interacted with each other to meet each other needs[5,6]. The process of making relationships is so-called IPC and it is provided different strategies for it so far which discuss about it later in this paper. Figure 1 indicates the concept of IPC.

3. The Previous Strategies about IPC

There are different mechanisms to make relationships among the processes which have advantages and disadvantages.

File System IPC method is a mechanism in which the process writes the source of data in a file and then reads the data target from the file. It has a simultaneous problem but it can be removed by locking. Also, by using a specific type of files, the using method can be facilitated[5,7]. The other applied mechanism is Message-Based IPC. In this method, the process puts the data source in a

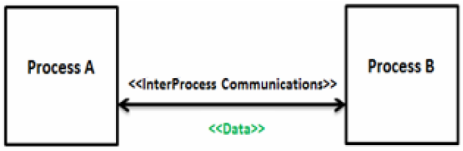

Figure 1. The Relationship between Process A and Process B.

specific frame in a message and sends it to the OS. Then, the OS places the message in the array of input messages of target process. Finally, the target process reads the message from input array. In this method, the message sender can be either waited the receiver reply or not[8]. The other applied method is procedure call IPC method which is used sub-procedures to make relationship among the processes. The data are sent through parameters and the reply reaches to the recipient as a returning value. The process usually waits for sub-procedures reply and in fact, it is blocked. As the target sub-procedures put in a different address space, the complexity is increased[8,9]. Shared Memory IPC is another mechanism to make relationships among the processes. In this method, the different processes share a general part of memory among themselves. This memory can be either physical or available virtual. The relationship among these processes is provided through this Shared Memory reading and writing. It is necessary to use mechanisms to remove the simultaneous problem and/or apply semaphore[10,11]. The issues which must be considered important about IPC and most available methods have deficiencies about it include the simultaneous problem among the processes to the variable or the Shared Memory during access time, the blocking of source and target processes after message sending to receive the reply from the other party and also the addressing method in IPC. Finding an optimal method to solve these problems will help us to reach a mechanism with high efficiency to provide secure and fast relationships among the processes[11].

4. Proposed Methods

Due to our proposed architecture about the OS and processes function, in this paper, we want to divide the system architecture to two separated parts with different functions. The necessity of providing two separated parts in the system indicates that the nature of expandable systems involve parts in which handle general and common tasks among all system parts and in fact performs the management tasks of total system. The structure of these parts in the system is almost constant and its goal is to manage variable components in the system. It is so called system core. Beside the core, there are other parts in the system with variable availability and can act according to different goals in the system. The activity of these parts is done under the control of system core so the elements placed in this part are so called module.

We consider the OS as the central core which in fact handles the management tasks in the system and the processes as modules which do their tasks under the control of the available processes in the system. The OS handles the main tasks as the system core as it knows about the available status of the processes and identifies and controls them. Consequently, by designing an Event Bus which is capable of making relationship with the available process table, it can be provide relationship and interact among different processes. The processes are also can be structured as the separated parts of the OS involved a connector besides the independency feature which is capable of interaction with each other through the available Event Bus in the system core[12,13]. Figure 2 indicates the system general frame and its relationship.

As it can be seen in Figure 2, the space of the OS is divided to two separated parts of Kernel Space and User Space. The Kernel Space plays its role as a central core in our proposed system and includes two parts: Process Table and EBCP. The Process Table can be used as a reference for Event Bus to access and identify the processes. Event Bus or EBCP is acted as an intelligent pipeline in the system and is accessible in the total system and all processes. EBCP as a general connector is capable of making relationship among the system different parts and processes. It processes the messages based on standard structure and then identifies it by using Process Table of the system processes and/or target processes. Then, it stimulates the message event in the total system and distributes it.

In the other part of the OS, there is User Space which includes the available and in process processes of the system which can be structurally different[14]. However, each one must have a standard and defined connector to make relationship with the internal structure and other processes. The Event Engine can be produced by the process itself but it is better to produce by the OS due to its

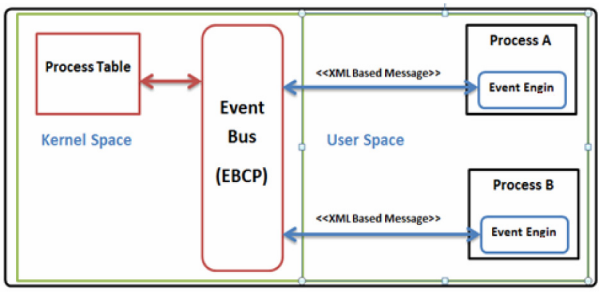

Figure 2. The General Structure of the System and its Interaction with Different Parts to make Relationship among the Processes.

specified internal structure. Then, during the process entry to the memory, it delivers to the process. The Event Engine connector acts as an interface between the process and external environment. It receives and sends the message using method based on message and XML structure. The Event Engine connector of each process has mutual interaction with the available EBCP in the core of OS. In fact, each action or reaction from system processes and/or core must be done through EBCP. As a process decides to make relationship with another process, firstly, it produces a message which includes receiver and sender characteristics as well as the message major data in a structural frame based on XML through its specific Event Engine. Then, the Event Engine process sends the message to EBCP. After receiving message, EBCP processes it and identifies the target processes using Process Table and finally by using an event, distributes the message among the target processes, simultaneously. The target processes produce a message with similar structure and send to the source process if replying to the message is required[15,16].

4.1 The Structure of the OS for the Processes

In our proposed strategy, the OS must have a series of distinct parts to control independent processes and manage them during the availability of the process in the memory. As some of these controlling features are essential for accurate function of processes and also for the OS as the manager of processes, so, these strategies are provided in the OS before[16]. So, we use these controlling features as a key to reach the processes in our proposed strategy. The OS uses process table to control and manage the processes. Consequently, it is used Process Table as a reference to access the available processes in the system and making relationship between the source and target processes[14,16]. The available EBCP in the OS which handles the relationships among the processes is so called EBCP. Its internal structure is indicated in Figure 3.

As it can be seen Figure 3, the Process A produces a message based on XML structure and sends it to EBCP of the OS to make relationship with A, B and C processes. The sent messages to EBCP are firstly stored in input buffer. It causes that as the received and sent messages of processes is increased, EBCP can manage them better. The available messages in input buffer are processed as first in first out by Event Processor Engine, respectively.

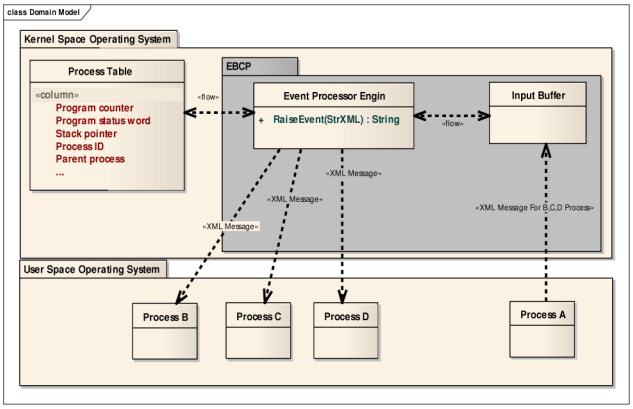

Figure 3. The Internal Structure of EBCP and the Relationship of Internal and External components with each other.

Based on determined policies for Event Processor Engine module, it sends the available messages in input buffer to all or some of system processes. This module has a behavior called Raise Event which handles message distribution in the system[17]. EBCP module uses process table as a reference to access and identify the processes.

In this step, it must be considered a basic and important problem in efficiency and security of system and that is the distribution method of sent message from a process[17,18]. When a process makes a decision to communicate with other process, it must produce a message involved necessary data in a pre-defined structural frame and then sends it through its connector to EBCP. After message entry to EBCP, it must be determined the system policy about how to deal with messages to provide the system security and efficiency in the best possible way[19]. In this case, the OS can be acted in two ways:

i. the OS can act in a way to generally distribute the received message of a process in total system and sends it to all available processes in the system. In this case, there is no need to add additional data field to process table. All available processes receive the sent message system from a process. The advantage of this case is that the message structure is simpler and decreases system complexity. There is also no need to add additional data field in process table. It can be noted to the system security decrease as the disadvantage of this case. Because, sent message become available for all system processes via a process and this may not be desirable for the source process. Although, can be used coding methods to find a method of data

access controlling. The other method of this case is the additional load which applies on input buffers of system processes and resulted in receiving the sent messages of system other processes. It is as most messages may not be useful for the system and practically consider as spam[19,20].

ii. in another case, the OS can act in a way in which as a new process enters the system, the OS must provide circumstances between in process and newly-entered processes. It causes that the processes which related and interacted, can find each other. Then, the newly-entered process identifies the authorized processes of interaction and provides their data for the OS. Then, the OS records the received data as well as other data of newly-entered process in the process table. In fact, it is required to save additional data in the processes table[20]. Figure 4 indicates the stages of operation as sequence diagram.

Due to Figure 4, as a process enters a system, the OS records the process in the process table after assigning variables and required space to the process as well as specific Event Engine. Then, the OS sends a message containing an ID of newly-entered process to each available process in the system. As soon as the available processes in the system receives the message from the OS, it sends a cooperation request message to the newly-entered process (if needed) which includes a key that only friend and cooperation processes can identify and process. The newly-entered process reviews the received key from other process after receiving cooperation request from other processes of the system. If the key is accurate, the considered process will be adopted as the cooperation process and asks for the OS to record the noted process as the cooperation process in its specific entry in the processes table[21].

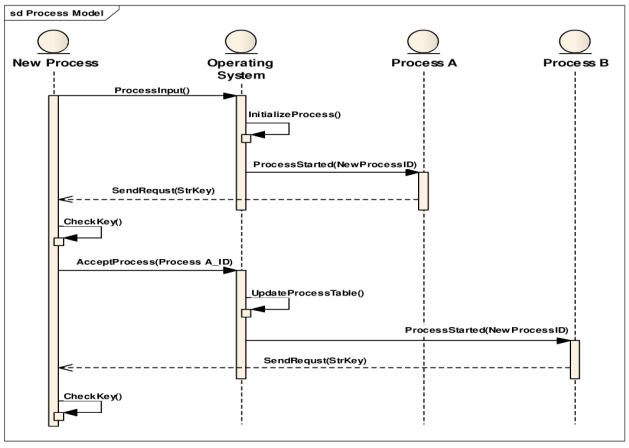

Figure 4. Sequence Diagram of Interaction among the Processes, OS and Newly-Entered Process to Collect Related Processes.

It can be noted to the full security of relations among the processes as an advantage of it as only allowed processes which are validated previously receives the sent message. In this case, there is no additional load on the input buffer of the processes because the characteristics of related processes in the process table and additional fields which is provided to define the related processes are available. So, the OS uses these data and generally distributes the message among the related fields to the message source. As a result, the message won't be sent to the other available processes which don't have relation with the system source and their input buffer will be empty of useless and redundant message. In this case, their input overload will be considerable decrease. It can be noted to the weakness of this case which includes adding additional data field to the process table to determine the related processes and also input load of system which resulted in interaction among the processes during new process entry to the system[20,21].

4.2 Standard Connector Definition of the Process

After defining EBCP structure and its function about the received messages from the processes, we come to discuss about their structure and how they begin to make relationship with the other processes and which stage and steps must be taken to provide a proper and suitable relationship. As the available processes in the OS are organized and designed based on modern OS structure, so, changes in their internal structure and providing a standard structure for all of them will expensive and create a kind of limitation in the system[19–21]. We try to keep the current status of the processes internal structure constant and don't apply changes as the process acts as necessary in their internal structure. The important fact is the process external structure and hoe to interact with the other processes. Along this, to reach the goal, the best method is to define a standard connector of producing messages with identifiable structure in the system and sending to the available EBCP in the OS. Also, if a message is sent to the process from EBCP, the connector could receive, identify and process it[22].

The point which must be considered here is that to regard the structural independency of the processes and lack of limit for the processes, producing and defining the process connector isn't considered as the tasks of the connector and must be produced by the specific connector of the OS. The reason is that structure of connector for all system processes is constant and defined. As the connector must have a standard structure in total system, it is better that the OS produces a specific OS and specifies it to the processes. The method is that when a program is converted to a process and the OS records its data in the process table, the OS produces the specific connector of new process based on inserted characteristics in the process table and also available data about the connector structure and specifies it to the process. This connector is available in the whole cycle of available process living and will be accessible and identifiable via OS. When the process function ends up, the process connector is destroyed by the OS and the process is removed from the system. We called the process connector Event Engine to determine it from the main structure of the processes[21–23].

4.3 The Internal Structure of Process Connector Event Engine

As we noted in Section B, the OS as the process entered, specifies the specific connector of newly-entered process and then if it is necessary to have relationship with the external environment and special with the other process in the system, it uses specific connector Event Engine. But the important fact is the internal structure of the process connector (Event Engine) and how to function in different conditions. To reach the optimal and flexible structure in designing Event Engine, it must be considered the tasks the connector handles about the process and the system[23]. The process task, due to the system expectations from processes and the processes from each other, can be changed. So, dynamic extensibility is an important factor which must be considered in designing Event Engine. But the most important and main task of Event Engine is to send and receive the related messages to the process as it makes the process capable of making relationship with the system and other processes[23,24]. If we want to explain the relationships among the processes in detail, it can be noted to the cases such as producing the sent messages of the process, coding and decoding of messages which Event Engine handles. Due to the noted points, Figure 5

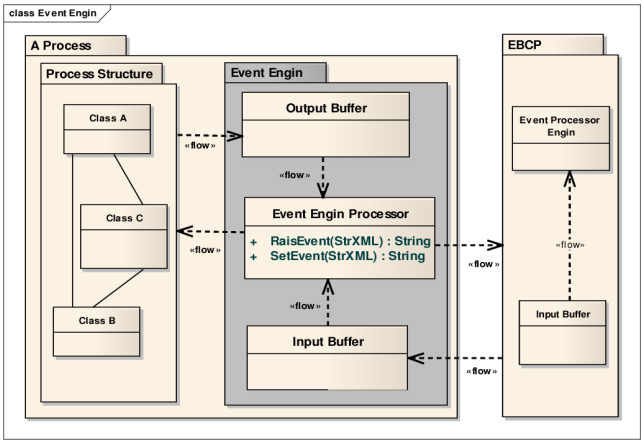

Figure 5. The Internal Structure of Process Connector Event Engine and How to Make Relationship with the Other Parts.

indicates the proposed model of Event Engine internal structure in which the edibility and flexibility as quality features are considered in it.

As it can be seen in Figure 5, Event Engine acts as a module inside a process based on the process structure. The Event Engine consists of two input and output buffer. The goal of designing Event Engine is based on buffer is that it keeps the available Event Engine efficiency and reaction in the connector when there is high traffic in the system and the process is capable of processing and answering to all messages.

The output buffer includes received messages which from the system core and/or other processes. The input buffer includes received messages from the system core and/or other processes. Event Engine Processor is responsible to distribute available messages in input and output buffers. As soon as a message delivers to the input and output buffers, EEP indicates reaction and sends the message to the target after coding or decoding[25].

4.4 The Structure of Sent and Received Messages

We use an architecture based on event to distribute and send the messages in the system. So, it concludes that our method to communicate and make relationship between processes and different parts of the system is based on message passing method[26]. We must answer to a question that how must be a message which includes sent and received data between processes and the system designed to known as integrated in total system and can be processed by all processes? It is clear that a designer can

improve the system efficiency using different methods and define the message structure.

In designing the message structure, it must be considered points such as the message must keep the main data in the best possible way and completely readable for the target destination. The structure of the message must be in a way to determine completely the source and target of the message[26]. So we use the well-defined structure of XML to design message structure. The goal of designing the message structure based on XML is that we can maximize the readability of the message by defining particular tags of different parts of a message and consequently increase the message processing rate desirably by processes. Developing and improving the structure of the message to reach the determined goals in the system is also easily implemented.

In Figure 6, you can see the sample of sent and received messages structure.

The characteristics of each tag in the provided structure are provided in Figure 6 and their usages are shown in Table 1.

5. Reviewing and Evaluation the Proposed Method

The processes frequently need to make relationship and communicate with each other and there are different methods to do this but we prefer to do this in a way which is better and more structured than using pauses. In fact, in providing a strategy, it must be considered several points[7,10]. Firstly, how can two processes exchange data with each other? What can we do to make sure that two or more processes don't interfere in their critical activities? When there is correlation between two or more processes, how can we perform the synchronization among

```
<Event EventType="Sent | Reply" EventID="ID" SentEventID=""
SenderName="senderName" SenderID="senderID" ReceiverName="Core | a Process | *"
ReceiverID="ProcessID | *" DateTime="">
        <InputFields>
                <Field Name="fieldName" Value="value">
                ...
        </InputFields>
        <OutputFields>
                <Field Name="fieldName" Value="value">
                ...
        </OutputFields>
</Event>
```

Figure 6. The Structure of Exchanged Messages between Processes and the OS Core Based on the Proposed Method

Table 1. The provided tags in the available message structure in Figure 6 and their usage

Tag	Description
Event Type	Determines that the message is sent or delivered in reaction to the sent message
Event ID	Determine the event and/or message code and can be unique and controlled by the system core
Sent Event ID	If the message is returned one, it determines that the message is sent in response to which sent message
Sender Name	Determine the name of process or the message sender which can be the OS core or a process.
Sender ID	Determine the process code or the message sender unit and can be inconsistent with the available processes table in the OS.
Receiver Name	Determine the name of recipient which can be a unit of the OS core or one or a few process.
Receiver ID	Determine the recipient and can be arranged based on the processes table.
Date Time	Determine the time and date of message delivery
Input fields	It includes data in which the source sent as input to the target or data in which the source request the target. The data of this feature can be a series of ID and values.
Output Fields	It includes data in which the target determines as the message reply after processing and returns it to the message sender. The data of this feature can be a series of ID and values.

the processes to increase the efficiency? Our method to send data is based on message delivery as it organizes the data in XML structural frame and exchanges among the processes. The defined structure for the messages is in a way to preserve the main indicators of each message and can add or remove different controlling options in the future to it if necessary without its affection on processes and other OS parts. It is because of the dynamic nature of XML structure and its support from different kinds of protocols of data transfer and preserving their security using coding algorithms which provide data transfer to remote machines through heterogeneous networks. It is as the previous methods such as Procedure Call, the data transfer is done through parameters in which their numbers and types are static and decrease the ability of its change and develop[7,8].

It can be said about the simplicity of implementation that Message-Based, Shared Memory and File System methods have less complexity in implementation than

Procedure Call method if there is different addressing space. However, these methods are all related to the addressing space and must perform additional operations and controls whether the addressing space is same or different[8,9]. While in the proposed method, the required operation to interact between two processes is related to the minimum data of the environment and in fact independent from circumstances two processes have to each other which considerably decrease the complication. File System and Shared Memory methods still faced with synchronization problem among the processes although strategies such as locking is provided to sole it but these strategies increase the complication and decrease the efficiency[8,10,11]. While our proposed method which compound Message-based method and software architecture based on event not only keeps the positive features of Message-Based using events which acknowledged source and target processes during occurrence but also increase the intelligence and synchronization of processes. In Table 2, it is indicated a general comparison between our proposed method and the previous ones. The supportive amount of these methods also indicates the required features for an optimal IPC.

6. Conclusion and Future Works

In this paper, we provide a strategy to make combination of IPC based software architecture based on event and communication method based on message passing. As by putting an EBCP in the OS core and also using OS processes table as a reference to access to the available processes in the system, we use it as a communication bridge among the processes. Moreover, to integrate processes with the available EBCP in the OS, we use a connector for each available process in the system. Providing relationship between the processes and available EBCP in the OS which considered as a communicational highway among the processes is the main tasks of a connector. We also provide a standard frame based on XML for received and sent messages structure to increase the readability and processing rate of messages. It can be noted to the main features of the proposed method as the lack of synchronization among the processes, implementation simplicity, easy expansibility and remote access.

Due to the above-mentioned points, the proposed method can be used as reference to communicate between processes and also processes with OS of different kinds and structures. It can also be used message coding and decoding methods to maximize the security of exchanged messages among processes. It considers not only the private limits of modules but also guarantees the communication security among them. It can also take major steps by assigning main responsibilities to the processes at their structural independency and increase the OS modularity.

7. References

1. Theodor AL. Operating system structures to support security and reliable software. Institute for Computer Sciences and Technology, National Bureau of Standards, Washington D.C; 1976 Aug.
2. Swift M, Bershad B, Levy H. Improving the reliability of commodity operating systems. ACM Trans. on Operating Systems. 2005; 23:77-1..
3. Goldberg RP. Architecture of virtual machines. Proc of the Workshop on Virtual Computer Systems, ACM; 1973. 74-112.
4. Cox A, Mohanram K, Rixner S. Dependable ≠ unaffordable. 1st workshop on Architectural and system support for improving software dependability; 2006; San Jose, California.
5. Grimshaw A, Humphrey M, Knight JC, Nguyen-Tuong A, Rowanhill J, G. Wasson, Basney J. The development of dependable and survivable grids. 2005 Workshop on Dynamic Data Driven Applications, Emory University, Atlanta, pp. 22-25, 2005.

Table 2. Comparing the proposed method and the previous ones based on the features of an optimal IPC

IPC Mechanisms	Synchronization	Works Remotely	Extendable	Easy to Implement
Shared Memory	Weak	Weak	Weak	Good
File System	Weak	Weak	Weak	Good
Message Based	Good	Good	Good	Weak
Procedure Call	Good	Weak	Weak	Weak
Proposed Solution	Excellent	Good	Excellent	Good

6. Nicol DM, Sanders WH, Trivedi KS. Model-based evaluation: from dependability to security. IEEE Trans Dependable and Secure Computing. 2004; 32:1–3.
7. Avizienis A, Laprie J-C, Randell B, Landwehr CE. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, Vol. 1, pp. 11–33, 2004.
8. Candea G, Kawamoto S, Fujiki Y, Friedman G, Fox A. Micro reboot – A technique for cheap recovery. Symposium on Operating Systems Design and Implementation; 2004; San Francisco, CA . p. 31–44.
9. Tanenbaum AS, Herder JN, Bos H. Can we make operating systems reliable and secure? Computer. 2006; 39: 44–51.
10. Garlan D, Shaw M, Okasaki C, Scott C, Swonger R. Experience with a course on architectures for software systems. Proceedings of the Sixth SEI Conference on Software Engineering Education; 1992 Oct. Springer-Verlag. LNCS 376.
11. Randell B. Operating systems: the problems of performance and reliability. University of Newcastle upon Tyne, Computing Laboratory; 2007.
12. Bao Y, Sun X, Trivedi KS. A workload-based analysis of software aging, and rejuvenation. IEEE Trans Reliability. 2005; 54:54-57.
13. Fowler M. Patterns of enterprise application architecture. Boston: Pearson Education; 2003.
14. Fähndrich M, Aiken M, Hawblitzel C, Hodson O, Hunt G, Larus JR, Levi S. Language support for fast and reliable message based communication in singularity OS. Proceedings of the EuroSys 2006 Conference, Leuven, Belgium; 2006. p. 177–90.
15. Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000. IEEE Standards Department. The Architecture Working Group of the Software Engineering Committee; 2000.
16. Riehle D. The economic motivation of open source software: Stakeholder perspectives. IEEE Computer. 2007 Apr. 40(4): 25–32.
17. Pope K. Zend Framework 1.8 Web Application Development. PACKT Publishing; 2009.
18. Ahn G, Hu H, Jin J. Security-enhanced OSGi service environments. IEEE Trans Syst Man Cybern C Appl Rev. 2009; 39(5).
19. Fowler M, Rice D, Foemmel M, Hieatt E, Mee R, Stafford R. Patterns of enterprise application architecture. Addison Wesley; 2002.
20. Hall RS, Pauls K, McCulloch S, Savage D. OSGi in Action. Manning Publications; 2011.
21. Chappell D. Enterprise Service Bus. O'Reilly Media, Inc; 2004.
22. Bos H, Samwel B. Safe kernel programming in the OKE. IEEE Open Architectures and Network Programming. 2002; 141–152.
23. Chou A, Yang J, Chelf B, Hallem S, Engler DR. An empirical study of operating system errors. Symposium on Operating Systems Principles; 2001. p. 73–88.
24. Herder JN. Towards a true micro kernel operating system. Master's Thesis. Vrije Universiteit Amsterdam; 2005.
25. Chen D, Dharmaraja S, Chen D, Li L, Trivedi KS, Some RR, Nikora AP. Reliability and availability analysis for the JPL Remote Exploration and Experimentation System. Proc. of DSN; 2002.
26. Kim DS, Machida F, Trivedi KS. Availability modeling and analysis of a virtualized system. Proc. of PRDC; 2009.