The Design and Implementation of Fault Tolerant PSTR on the Embedded Virtualization System

Jinho Yoo

***Abstract*— This paper is related to design and implementation of the application based on a PSTR model on a virtual machine. We will load one hardware system with two virtual machines to implement the PSTR model. We will conduct a research about the configuration of a management module, sharing of serial device simultaneously, and communication between tasks on different virtual machines. The management module can be located on the host operating system or on another virtual machine. This paper will decide the position of management module by comparing the performances of management modules in two different positions. Then, this paper will analyze the task execution environment on a virtual machine related to I/O device sharing and communication method between application tasks. After analyzing, we design and implement the modified virtual port suitable for the PSTR model. We will also make inter task communication features on different virtual machines and compare the performances of this communication features with their experimental results.**

***Index Terms*— virtual machine, task communication, device virtualization**

I. Introduction

THIS paper will implement the application on a virtualization of embedded system[1]. This application is based on PSTR task model and uses serial ports. The PSTR model on this application consists of the primary and shadow processes. We will make a virtual hardware system, and implement the PSTR model process on this virtual hardware system. Application tasks use their serial ports for hardware input, and tasks, on different virtual machines, communicate with each other. We must solve the problems which occur when we migrate the features of real system into virtualization system. We also have to solve the problems which result from concurrent execution. We need to make several additional software modules for virtualizing hardware features. First one is the management module which requires to supervise the overall system. Second, as a system requirement, we must make possible of the simultaneous use of serial ports for the PSTR implementation[2]. Also, those serial ports on the virtualization system have to satisfy the performance requirements of the hardware serial ports. Lastly, a communication method between the primary and the shadow virtual machine has to be provided. This paper will implement and modify system software components to solve these problems which will be supported by experiments further discussed.

Jinho Yoo is with Baekseok University, South Korea (phone: +82-41-550-0493; e-mail: yoojh@bu.ac.kr).

II. Related Research and Background

A. *Related Works*

The virtualization technique has been applied to many hardware system to improve its availability. There are a lot to consider over the implementation of the virtualization because the hardware system has many different components. So, there are many researches on virtualization and we are interested in the implementation of devices and their performances among these researches. There are researches on the performance delay of the device I/O which shows that software systems can perform as well as hardware systems[3]. There are studies that show how I/O device affects the performance of the whole system and the implementation of timeliness[4].

B. *Research Background*

As presented in Fig.1, the PSTR model, in accomplishing a task, shadows the actions of the primary. The shadow exists so that when an error occurs in the primary, the shadow takes over the job while the primary recovers. The primary and the shadow need to communicate in order to synchronize their data for checkpoint.

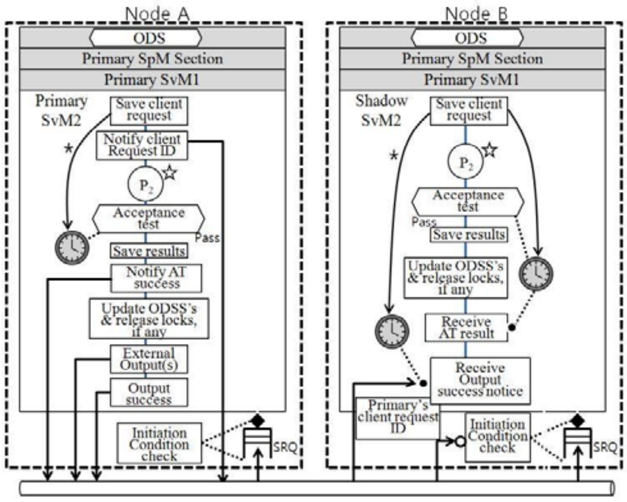

Fig. 1. PSTR Model

ISBN: 978-988-14047-1-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The clock in Fig.1 shows the real time clock for the real time requirement. The asterisk denotes the ODS(Object Data Store) which represents the global variables.

The primary saves the client request and notifies the shadow of the client request ID. Then P2 is executed, which is followed by an acceptance test. The shadow executes the same processes. In order to check the validity of the primary, the shadow synchronizes with the primary.

To support the implementation of the PSTR model, we will propose the virtual serial ports required for hardware input, and the direct communication paths between application tasks. This paper will also implement a virtual channel that has no transmission delay between application tasks on different virtual machines.

III. SYSTEM IMPLEMENTATION

The application program of the PSTR model has been implemented on general hardware system, and in this research, this implementation will be done on virtual machine, which supports full virtualization. Also, this research will find the most efficient ways for virtual machines to meet hardware system's performance criteria and implement the PSTR model on virtual machines.

The tasks are executed on hardware, and they use the socket feature to communicate with each other. Also, the hardware system provides general basic I/O, such as serial port. In this research, the tasks will be executed on virtual machines, and perform direct channel communication, without using socket interfaces. Also, this has to be able to control the overall hardware component which is implemented by software techniques. Thus management module is needed to control the overall virtualization system. In addition, this management module will manage to implement PSTR model on virtual machines that are capable of such features.

The hardware target system for this research's implementation and its corresponding performance evaluation is consisted of a processor that is AMD 64bits, 2.0 GHz, and a 8G RAM. The Operating System is a real-time patched Linux Kernel 3.4.0.

A. *Management Module*

There are two ways to implement a management module. The first way is to make management module a process on a host operating system. This way is the most efficient way in terms of communication performance. However, if the host operating system is changed, the management module has to be made again in a different host API environment. The second way is to make a virtual machine for the management module.

This is very portable because it only needs the virtual machine to be executed on a host operating system. However, this way will take more time than the first way because of the overhead of virtual machine[5].

This research will choose between a good communication performance and portability. This system prioritizes speed over other features, so, we will experiment to find out and confirm which way is faster in a real system.

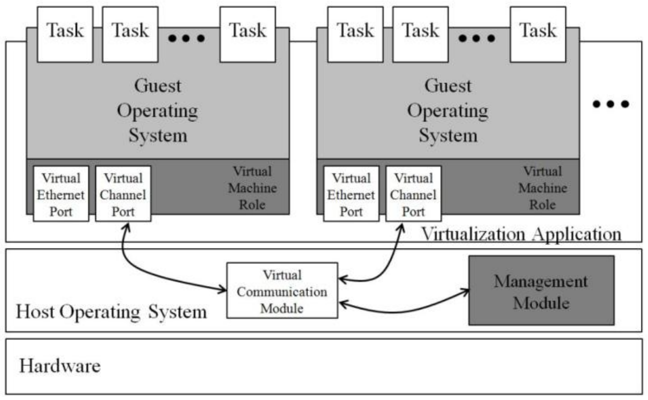

Fig. 2. Management Module on Host OS

B. *Communication Performance*

This research will compare the performance of the management module implemented as a host process, and the management module implemented as a virtual machine in Fig 3. The horizontal line represents byte scale transmitted and the vertical axis represents the duration of transmission. Implementing the management module on a virtual machine is denoted by a dotted line, and implementing management module as a host process is denoted by a solid line. Implementing management module as a host process is faster in most case. Implementing it on a virtual machine is slower because the transmitted packet has to pass through the operating system layer on the virtual machine. We chose to make the management module a host process in terms of communication speed that is shown in Fig.3.

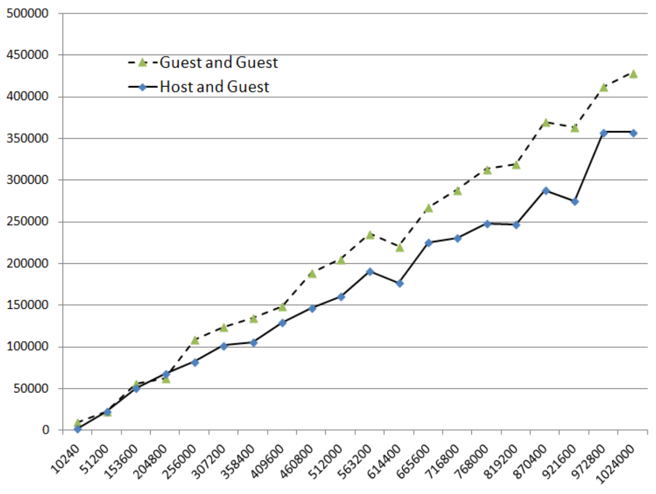

Fig. 3. Communication performance

C. *Sharing Serial Port*

There are primary virtual machine and shadow virtual machine on a host system as shown in Fig.4. Two virtual machines share the serial port of host system as shown in Fig.4. If the data is entered to the serial port of the host system, Distribution Module Process copies and distributes the entered data into the I/O redirection stubs of the primary and the shadow virtual machine. Then, the primary and the shadow virtual machine each will have the duplicated data. Distribution Module Process receives the entered data from host serial port and puts it into the input buffer. I/O Redirection Stub will get it from the input buffer. For output, I/O Redirection Stub puts the output data into the output buffer of the Distribution Module. The primary task reads the data from virtual serial port and writes the output data into virtual serial port. Management Module and Distribution Module are responsible for delaying and queuing of input and output related to host serial port. They maintain time-stamped data for resolving timing problems.

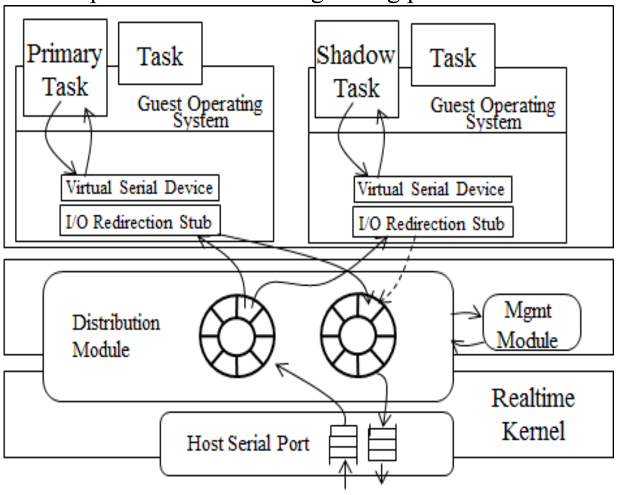

Fig. 4. Serial Port Sharing

D. *Performance of Virtual Serial Port*

Virtual serial port is the data structure managed by a software program module and has to satisfy performance standard of the serial port of hardware. Therefore, virtual serial port has to meet 115k bit per second. Sharing the hardware serial port may affect the performance of virtual serial port. So, this research conducted a test to know whether the performance meets the hardware performance criteria. The results is shown as Fig.5.

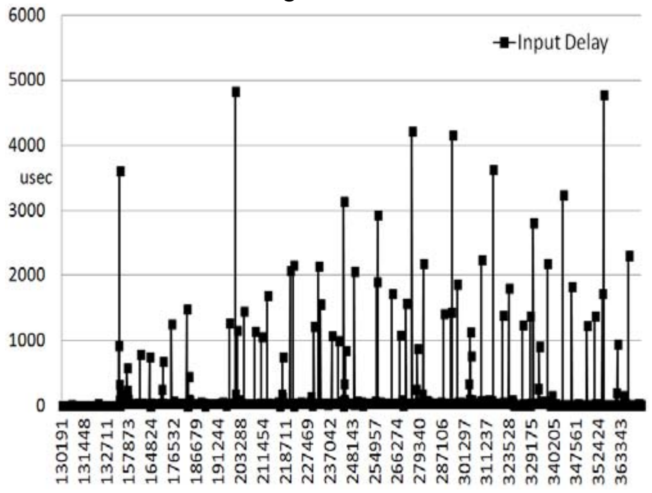

Fig. 5. Input Delay of Sharing Device

The horizontal axis of Fig.5 is the value of incremental real time clock. The vertical value represents the delay value to the clock. The time delay in Fig.5 is shown from 2 us to 5 ms. The I/O maximum speed of serial port is 115 kbps, that is, the I/O delay of transmission is approximately 70 us. This research added the performance control feature to make serial I/O constant speed.

E. *Direct Communication Path between Tasks*

This research designs and implements a virtual communication channel to support direct communication between application tasks on different virtual machines shown as Fig.6. This virtual network device is only used for communication between applications. This virtual network device exists only for software execution environment, not for real hardware environment[6]. This virtual network device has the address made from UUID(Universal Unique IDentifier) and task ID. In the process of transmitting the packets, this virtual network device supports timestamp function to record the time the packet was made.

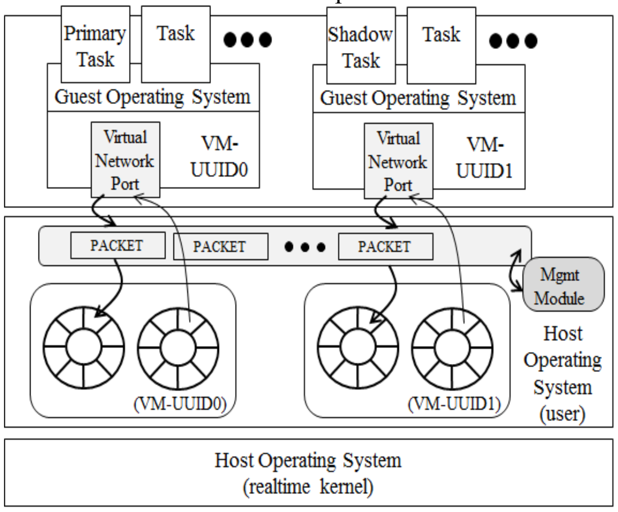

Fig. 6. Communications between Tasks

This virtual network device calculates the waiting time taken to get to the destination of packet. The phases are divided into Generation Phase(G_i), Queue Phase(Q_i) for queuing, Process Phase(P_i), Transmission Phase(T_i) for managing the time delay. Application tasks ask for simple deadline check mechanisms, which are "hard" and "soft". In case of "hard," the virtual network device decides whether the required time is greater than the calculated waiting time. If their packet is not able to be arrived at the destination, the virtual network device rejects the transmission from application task[5]. However, in case of "soft," the virtual network device just notifies the status to the application task.

$$MIN_{delay} = \frac{1}{N}\sum_{i=0}^{N}(MIN(G_i) + Q_i + P_i + MIN(T_i))$$

It calculates average delay time as the below.

$$AVG_{delay} \frac{1}{N}\sum_{i=0}^{N}(G_i + Q_i + P_i + T_i)$$

It compares RTT and AVG as following on the packet generation phase.

```
if ((RTT(packet)<AVG(delayDB))
    Packet is allocated from pre-allocated queue pool
    (MIN(G_i))
else
    Packet is allocated from memory
```

On the packet transmission phase, it compares as following

```
if ((RTT(packet)<AVG(delayDB))
    Packet is placed in front of queue,
    Notify packet arrived using SIGIO
    (MIN(T_i))
    else
        Packet is placed in queue

    RTT: Requested Transmission Time,
    AVG(delayDB): average time of delay in delayDB.
```

If the parameter for time requirement is soft, the transmission is executed after the time requirement is compared to the average minimum delay time.

In this paper, we compared the transmission speed of socket communication, and three different lengths of messages on the virtual network device: 32, 64, 1024 bytes. The result is shown in Fig.7.

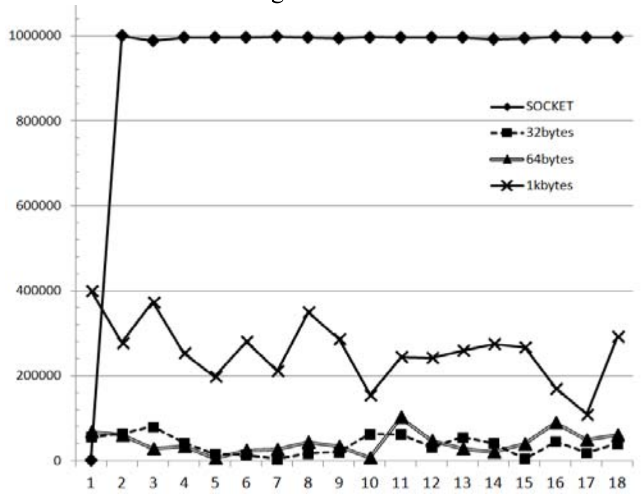

Fig. 7. Comparison of communication time

The socket communication shows the constant communication time if packet size is smaller than the constant size. This communication packet size is 1024 bytes per transmission, that is, this application just requires 1024 bytes size for transmission. The communication of the virtual network device of 1024 packet-size transmission is better than that of socket. Fig.7 shows that the proposed virtual network device is better than socket.

IV. Experiment and Demonstration

Experiment is divided into simulation and real experiment. We installed the virtualbox embedded with PSTR model for fault tolerant implementation. We used Flightgear software for the simulation and the quadcopter for the real experiment.

Demo scenario is as follows. Primary virtual machine executes its application tasks as its mission on sending heartbeat into shadow virtual machine periodically.

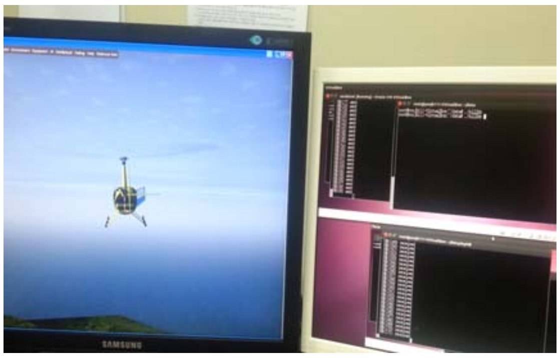
Fig. 8. Application Task on Primary Virtual Machine

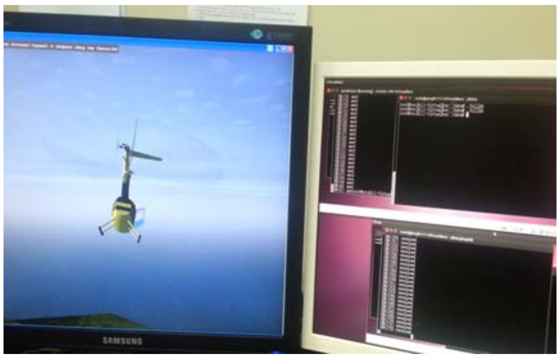
Fig. 9. Application Task Fail on Primary Virtual Machine

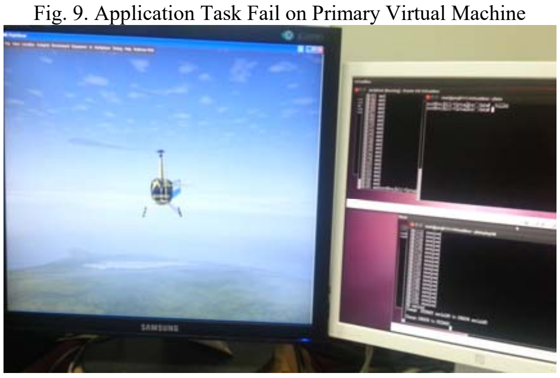
Fig. 10. Switch to Application Task on Shadow Virtual Machine

Shadow is monitoring the status of the primary side simultaneously. When errors occur on the primary, the shadow takes I/O control from the primary and continues the primary's mission instead. After that, the primary tries to recover itself. If it is recovered, it takes the I/O control and does its mission again. Fig.8 shows the snapshot of Flightgear software that the primary does his mission. The right side of snapshot shows the status of the virtual machine system. Fig.9 shows the system failure of the primary and once it fails, the helicopter declines. When the shadow recognizes the failure of the primary, the primary execution is switched into the shadow in Fig.10, and the shadow takes the responsibility of control.

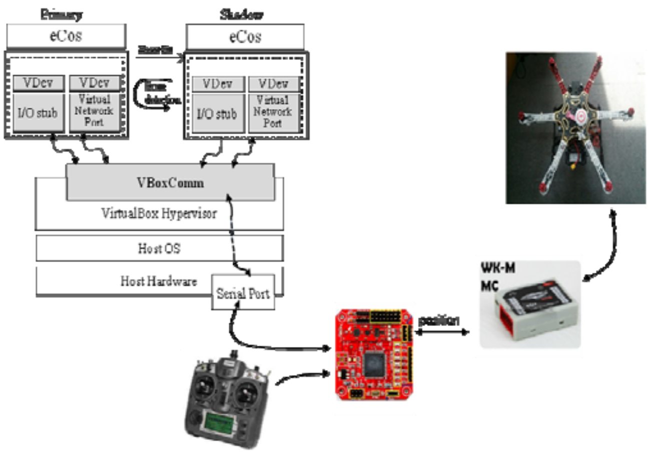

Fig. 11. Experiment Composition

In Fig. 11, there are a real experiment environment and software stack of real system. There are two virtual machines on a hardware system, and application tasks are on each of the virtual machines. The application task is executed on the eCos operating system, which is a guest operating system. The serial port and wireless controller both are connected to the flight control board using lines and these two lines control the quadcopter. We can be informed that the primary is switched to the shadow through monitoring feature.

V. CONCLUSION

This research implemented PSTR model for fault tolerant on a virtual machine. We modified and adapted the virtual machine(virtualbox) to implement the PSTR model as a embedded virtualization. We mainly modified I/O environment and made the private channel communication for direct transmission between the application tasks on different virtual machines.

We added the virtualization software module which supports the duplication, the control of I/O data, and flow controls. We implemented the virtual network device which is good for small data transmission, and we evaluated the performance of virtual network devices by comparing socket communication. We used Flightgear for simulation to verify the activities of the primary and the shadow, and tested this research on the quadcopter as a real hardware environment. In conclusion, this research succeeded in making the PSTR model possible of being efficient fault tolerant on an embedded virtualization environment by sharing serial ports, making direct communication and management module.

REFERENCES

[1] Ning Li, Yuki Kinebuchi, Hitoshi Mitake, Hiromasa Shimada, Tsung-Han Lin, and Tatsuo Nakajima, "A Light-Weighted Virtualization Layer for Multicore Processor-Based Rich Functional Embedded Systems", Proceedings of 15th IEEE International Symposium on Object/Component/ Service-Oriented Real-Time Distributed Computing, 2012, pp. 144-153.

[2] Zonghua Gu and Qingling Zhao, "A State-of- the-Art Survey on Real-Time Issues in Embedded Systems Virtualization", Journal of Software Engineering and Applications, May 2012, pp. 277-290.

[3] K. H. Kim, and C. Subbaraman, "The PSTR/SNS scheme for real-time fault tolerance via active object replication and network surveillance", Knowledge and Data Engineering, IEEE Transactions on, Vol.12, No. 2, Mar/April 2000, pp. 145, 159.

[4] Kim, K.H., Liu, J.J.Q., "Techniques for implementing support middleware for the PSTR scheme for real-time object replication," Object-Oriented Real-Time Distributed Computing, 2004. Proceedings. Seventh IEEE International Symposium on , vol., no., 14-14 May 2004, pp.163,172.

[5] Jinho Yoo, Kyujong Han, Yong-Hyun Kim, Mirim Ahn, Doo-Hyun Kim, “The Execution Environment Study of Primary Shadow TMO Replication Model on a Virtual Machine”, Korean Institute of Information Technology , vol. 11, No. 8, Aug. 31, 2013, pp.153-162.

[6] Nakauchi, K., Shoji, Y., Ito, M., Zhong Lei, Kitatsuji, Y., Yokota, H., "Bring your own network - Design and implementation of a virtualized WiFi network," Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th, vol., no., 10-13 Jan. 2014, pp.483,488.