INTEGRATION OF LINUX COMMUNICATION STACKS INTO EMBEDDED OPERATING SYSTEMS

Jer-Wei Chuang[ξ], *Kim-Seng Sew*[ξ], *Mei-Ling Chiang*[+], *and Ruei-Chuan Chang*[ξ]

Department of Information Management[+]
National Chi-Nan University, Puli, Taiwan, R.O.C.
Email: joanna@ncnu.edu.tw

Department of Computer and Information Science[ξ]
National Chiao-Tung University, Hsinchu, Taiwan, R.O.C.
Email: kssew@os.nctu.edu.tw, rc@cc.nctu.edu.tw

ABSTRACT

As the explosion of Internet, Internet connectivity is required for versatile computing systems. TCP/IP protocol is the core technology for this connectivity. However, to implement TCP/IP protocol stacks for a target operating system from the scratch is a time-consuming and error-prone task. Because of the spirit of GNU GPL and open source codes, Linux gains its popularity and has the advantages of stability, reliability, high performance, and well documentation. These advantages let making use of the existing open source codes and integrating Linux TCP/IP protocol stacks into a target operating system become a feasible and cost-effective way.

In this paper, we describe how to integrate Linux communication stacks into LyraOS, a component-based operating system for embedded systems. Under the component design principle, the communication stack should also be implemented as a separate and self-contained component. So the integration work should deal with the difference of system design principles and kernel architectures. This work includes modifying Linux communication stack codes and implementing LyraOS kernel support modules. Performance evaluation shows that for TCP transmission, LyraOS performed better than Linux by 7.39%. The experience of this integration study can be of practical value to serve as the reference for embedding TCP/IP stacks into a target system.

1. INTRODUCTION

Embedded applications are versatile and the hardware devices range from simple controllers to more complex systems. For the versatile hardware devices and different application requirements, a reconfigurable embedded operating system is needed. Thus, various operating systems design dedicated for embedded systems are thus created, such as PalmOS [22], EPOC [12], Windows CE [25], GEOS [16], QNX [23], Pebble [2,17], MicroC/OS [21], eCos [11], LyraOS [3-7,18,19,26], etc.

As the explosion of Internet, adding Internet connectivity is required for embedded systems. TCP/IP protocol [8,9,24] is the core technology for this connectivity. However, to implement the TCP/IP protocol stacks for a target operating system from the scratch is a time-consuming and error-prone task. Because of the spirit of GNU General Public License (GPL) [15] and open source codes, Linux [1] gains its popularity and has the advantages of stability, reliability, high performance, and well documentation. These advantages let making use of the existing open source codes and integrating Linux TCP/IP protocol stacks into a target operating system become a feasible and cost-effective way.

This paper describes how to integrate Linux communication stacks into LyraOS [3-7,18,19,26]. LyraOS is a component-based operating system designed for embedded systems. Under the component design principle [2,14,17,20], the communication stack should also be implemented as a separate component, such that the advantages of modularity, reconfigurability, component replacement and reuse can be maintained. However, there are many difficulties to deal with for this integration work. For example, being a monolithic kernel, Linux communication stack is not a separate component that has closely relationship and interaction with other kernel functions such as file system, device driver, and kernel core.

Therefore, the integration work should solve the difficulties from different system design principles and different kernel architectures. Our work focuses on two parts. First, implementing the communication stacks as a self-contained component, which requires modifying the Linux TCP/IP codes to separate them from other kernel functions. Second, implementing kernel support modules in LyraOS for integrating Linux TCP/IP protocols.

The rest of this paper is organized as follows. Section 2 briefly describes the difference between LyraOS and Linux, which affects the integration work. The difficulties that should be dealt with for this integration are also discussed and presented. Section 3 presents the integration work including modifying Linux communication codes and adding kernel support modules. Section 4 shows primitive performance evaluation results, and Section 5 concludes this paper.

2. INTEGRATION ISSUES

In this section, we first briefly describe the Linux communication stack architecture and the LyraOS architecture. Then the difficulties and problems encountered in the integration work are discussed and presented.

2.1 Linux Communication Stack Architecture

The basic Linux I/O system architecture is illustrated in Figure 1, which includes network subsystem and file system. The network subsystem includes socket layer, network protocol layer, and network device layer, as shown in Figure 2.

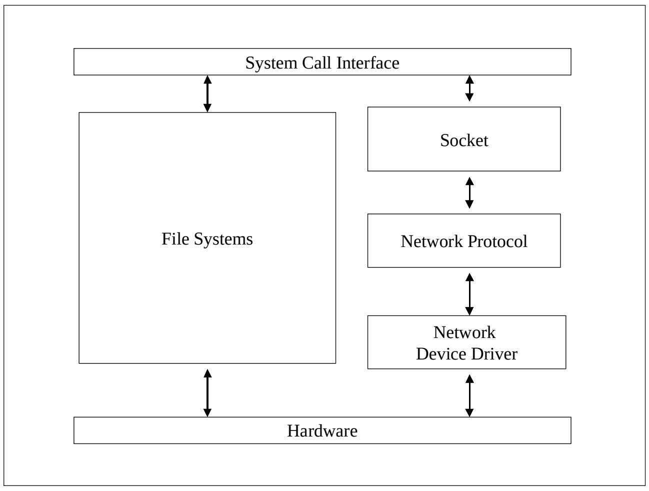

Figure 1: Linux I/O System Architecture.

Programmers make use of the socket interfaces to access network services. The invocation is made through system call interface in C library to enter into kernel's socket layer. The exported C library functions are listed in Table 1. Since Linux supports many different socket address families [1], so the main task of socket layer is to call the service functions of the requested address family (e.g. INET sockets). The INET layer will call the service functions of underlying TCP or UDP layers, which in turn will call the service functions of IP layer. The IP layer deals with the packets sent/received to/from network interfaces.

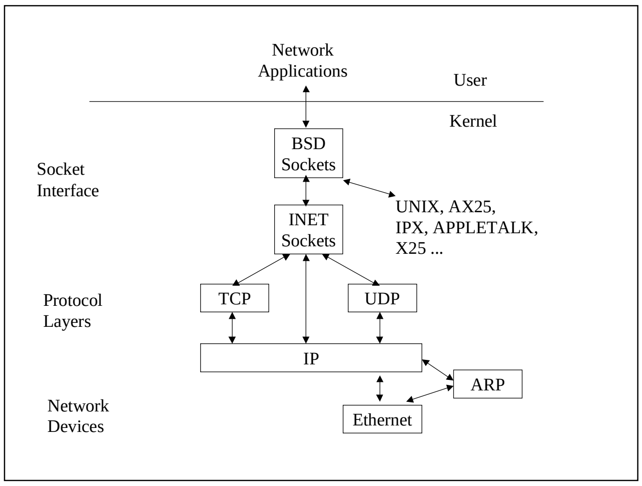

Figure 2: The Layer Architecture of Linux Network Subsystem.

```
int socket(int domain, int type, int protocol);
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
int listen(int s, int backlog);
int connect(int    sockfd, struct sockaddr *serv_addr, int addrlen );
int accept(int s, struct sockaddr *addr, int *addrlen);
int send(int s, void *msg, int len, unsigned int flags);
int sendto(int s,void *msg,int len,unsigned int flags,
                              struct sockaddr *to, int tolen);
int sendmsg(int s, struct msghdr *msg, unsigned int flags);
int recv(int s, void *buf, int len, unsigned int flags);
int recvfrom(int s,void *buf,int len,unsigned int flags,
                         struct sockaddr *from,int *fromlen);
int recvmsg(int s, struct msghdr *msg, unsigned int flags);
int getsockopt(int s, int level, int optname, void *optval, int *optlen);
int setsockopt(int s, int level, int optname, void *optval, int optlen);
```

Table 1: The Related C Library Functions.

Linux network subsystem shares some data structures and operations with file systems, which adds the difficulties in the integration work for implementing the network system as a separate component. As shown in Figure 3, when BSD sockets are used, kernel will create *inode* (consists of *socket*) and *sock* data structures and then make the corresponding links in the invoking process's *file descriptor table* and *open file table* [1].

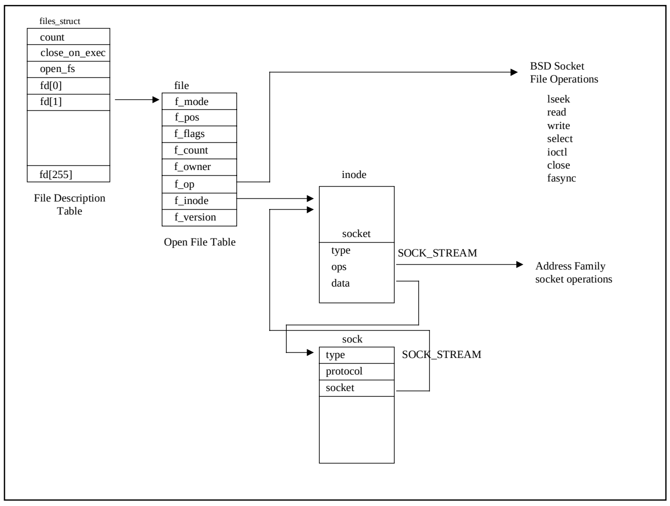

Figure 3: Linux BSD Socket Data Structures.

2.2 LyraOS

LyraOS [3-7,18,19,26] is a component-based operating system which aims at serving as a research vehicle for operating system and providing a set of well-designed and clear-interface system software components that are ready for Internet PC, hand-held PC, embedded systems, etc.

It was implemented most in C++ and some assembly codes. It is designed to abstract the hardware resources of computer systems, so low-level machine dependent layer is clear cut from higher-level system semantics. Therefore, it can be easily ported to different hardware architectures [4,6]. Each system component is complete separate, self-contained, and highly modular. So the system is also designed to be scalable and reconfigurable.

Besides being light weight system software, it is a time-sharing multi-threading kernel. Threads can be dynamically created and deleted, and thread priorities can be dynamically changed. It provides a preemptive prioritized scheduling and supports various mechanisms for passing signals, semaphores, and messages between threads. On top of the kernel core component, a micro window component with Windows OS look and feel is provided [18]. Figure 4 shows the system architecture.

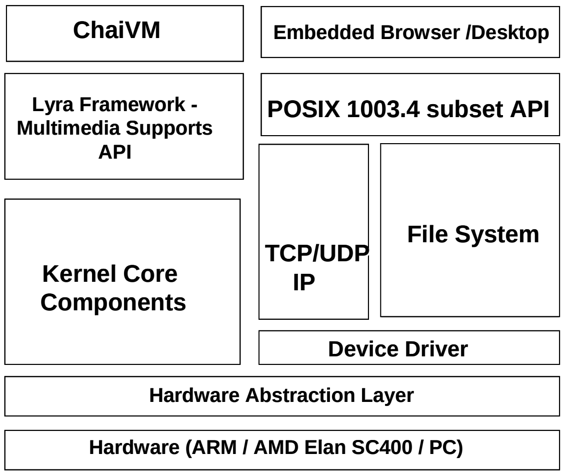
ChaiVM
Embedded Browser /Desktop
Lyra Framework - Multimedia Supports API
POSIX 1003.4 subset API
Kernel Core Components
TCP/UDP IP
File System
Device Driver
Hardware Abstraction Layer
Hardware (ARM / AMD Elan SC400 / PC)

Figure 4: LyraOS System Architecture.

2.3 Integration Issues and Difficulties

Since LyraOS and Linux are different in system architecture, Linux communication stacks must be modified for being integrated into LyraOS. LyraOS should also provide some kernel support functions for this integration.

Currently, LyraOS supports single address space [4,10] with static binding of applications and kernel. No system call invocation is needed for applications to use the kernel's exported services. To provide the compatible system call interface with Linux, LyraOS should provide the same socket interfaces for applications' use as Linux C library functions listed in Table 1.

Aside from the different OS architecture, under the component design principle, each LyraOS system component is complete separate, self-contained, and highly modular. Each component has clean exported and imported interfaces for components to communicate with. So, the communication stack should also be implemented as a separate component such that the advantages of modularity, reconfigurability, component replacement and reuse can be maintained. However, Linux is a monolithic kernel, its communication stack codes have closely relationship and interaction with other kernel functions such as file systems, device drivers, and kernel core. As introduced in Section 2.1, the socket layer shares the same data structures and operations with file system. Therefore, our first work is to separate Linux communication stack codes from file system codes.

To sum up, the integration work focuses on two parts. First, implementing the communication stacks as a self-contained component. We should clarify its import and export interfaces clearly and modify Linux TCP/IP codes to separate them from other kernel functions. Second, implementing kernel support modules in LyraOS for integrating Linux TCP/IP protocols.

3. DESIGN AND IMPLEMENTATION

This section describes the integration work including modifying Linux TCP/IP codes in Section 3.1 to 3.4 and adding kernel support modules in Section 3.5.

3.1 Socket Interfaces

As introduced in Section 2.1, programmers make use of the socket interfaces to use network services. This invocation is made through system call interface in C library to enter into kernel's socket layer. However, in LyraOS, no system call invocation is needed for applications to use the kernel's exported services.

To provide the compatible system call interface with Linux, we add in LyraOS the same socket interfaces for applications as in Linux C library functions as listed in Table 1. For example, the *socket* function called by applications is implemented in this way that it directly calls Linux socket layer service function as shown in Figure 5.

```
extern int sys_socket(int family, int type, int protocol);

int socket(int family, int type, int protocol)
{
            int err;

            NO_PREEMPT();
            err = sys_socket(family, type, protocol);
            PREEMPT_OK();
            return err;
}
```

Figure 5: socket Function.

3.2 Socket Descriptor

As introduced in Section 2.1, Linux network subsystem shares some data structures and operations with file systems, as shown in Figure 3. To implement the network system as a separate component, we modify Linux communication stack codes to separate them from file system codes as shown in Figure 6. It means there is no need to access process's *file descriptor table* and *open file table* for invoking networking services. So, when BSD sockets are used, kernel will create only *socket* and *sock* data structures and the address of *socket* data structure is the LyraOS *socket descriptor*.

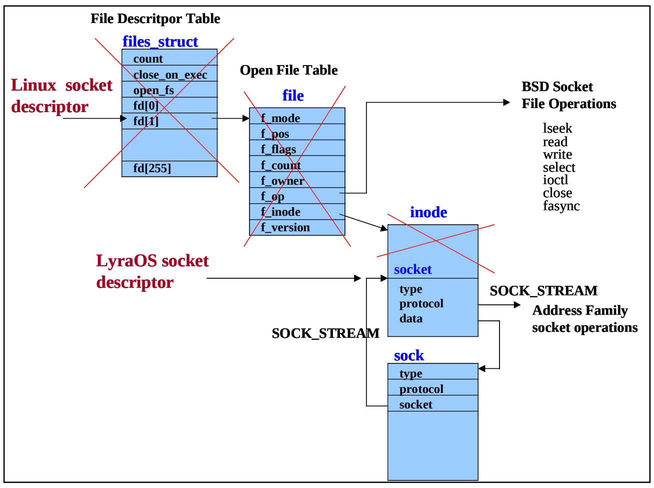

Figure 6: Socket Data Structures for Linux and LyraOS.

3.3 *select()* Function and *fd_set* Data Structure

In Linux, the *select()* function uses the *fd_set* data structure that is a bit array for mapping to *file descriptor table*. The *select()* function uses this bit array to check which corresponding socket is selected.

However, in LyraOS, this *file descriptor table* should not exist as explained in Section 3.2. Therefore, the *fd_set* data structure and its related manipulating macros should be modified. As shown in Figure 7, the *fd_set* data structure is modified to store LyraOS socket descriptor instead of bit array. It's related manipulating macros, i.e. FD_SET, FD_CLR, FD_ISSET, FD_ZERO, are modified to manipulate arrays of socket descriptors instead of bit array. Figure 8 shows Linux over LyraOS *fd_set* data structure.

```
#define __SELECT_FD_SETSIZE       63

typedef struct {
             int nr;
             void *fd[__SELECT_FD_SETSIZE];
} __select_fd_set;

typedef __select_fd_set fd_set;

#define FD_SETSIZE                __SELECT_FD_SETSIZE
typedef __select_fd_set       fd_set;
```

Figure 7: LyraOS *fd_set* Data Structure.

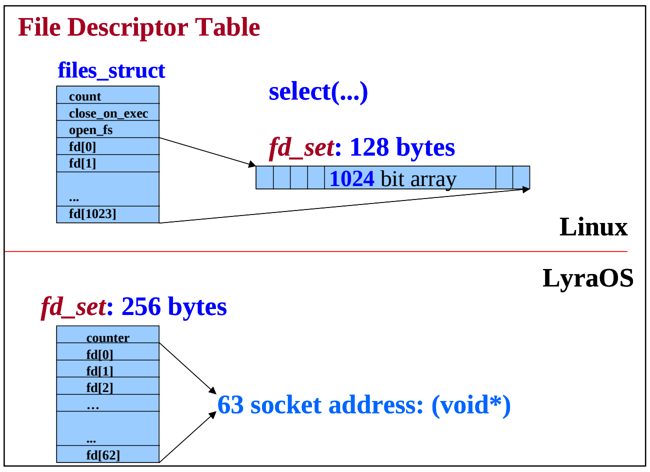

Figure 8: *fd_set* Data Structure for Linux and LyraOS.

Since *fd_set* data structure is modified to store array of LyraOS socket descriptor, the *select()* function is modified to manipulate the array of LyraOS socket descriptor instead of bit array.

3.4 Other Modified Communication Stack Routines

This section describes the other part of Linux communication stack codes modified for being integrated into LyraOS.

wait queue Related Functions

The Linux communication stacks implement the following wait queue related functions: *add_wait_queue(), remove_wait_queue(), sleep_on(), interruptible_sleep_on(), wake_interruptible(), wake_up()*, etc. To be integrated into LyraOS, these routines are modified to call LyraOS kernel's *thread_sleep()* and *thread_wakeup()* functions.

In Linux communication stack codes, the *current* system variable is used to represent the current running process. When porting to LyraOS, the related codes are modified to call *self_thread_id()* to get current running thread's id.

alarm() functions

In Linux, *alarm()* function can be used to set the timing information and timer related functions. Linux uses kernel's *add_timer()* function to implement the *alarm()*. Therefore, LyraOS kernel must also provide the timer related functions. The *alarm()* function is implemented as shown in Figure 9.

```
void do_alarm(unsigned long data)
{
        send_signal(data, SIGALRM);
}

int alarm(int t)
{
        static struct timer_list tl;
        tl.expires = jiffies + t * 100;
        tl.data = self_thread_id();
        tl.function = do_alarm;
        add_timer(&tl);
        return 0;
}
```

Figure 9: Implementation of *alarm()* Function.

schedule() Function

Linux communication stack uses kernel's *schedule()* function to relinguish current process's CPU execution. To be integrated into LyraOS, *schedule()* is modified to call the *thread_sleep()* that is a comparable kernel function to relinguish current thread's CPU execution in LyraOS. Figure 10 shows the implemented *schedule()* function.

```
void schedule(void)
{
        PREEMPT_OK();
        thread_sleep();
        NO_PREEMPT();
}
```

Figure 10: Implementation of *schedule()* Function.

sock_wake_async() Function

In Linux, *sock_wake_async()* function is used to wake up all processes waiting on the socket. However, this function is implemented through file system's operations and data structures. To be integrated into LyraOS and implemented as a separate component, this function is rewritten to use *wake_up()* function in LyraOS to wake up all the threads waiting on this socket, as shown in Figure 11.

```
int sock_wake_async(struct socket *sock, int how)
{
        struct sock *p;
        if (!sock)
                return -1;
        p = (struct sock *)(sock->data);
        wake_up(p->sleep);
        return 0;
}
```

Figure 11: Implementation of *sock_wake_async()* Function.

3.5 LyraOS Kernel Support Functions

This section describes what functions LyraOS kernel must support for integrating Linux communication stacks.

self_thread_id() Function

This function returns the current running thread's id that is the address of thread data structure in LyraOS.

thread_sleep() and *thread_wake()*

The *thread_sleep()* function used to relinghish CPU's execution is provided for the implementation of *schedule()* function and wait queue related functions as introduced in Section 3.4. The *thread_wake()* function used to wake up the waiting threads is provided for the implementation of *sock_wake_async()* function and wait queue related functions as introduced in Section 3.4.

Linux Device Driver Emulation Environment

Communication stacks must interact with network device driver for transferring packets from/to network interface card. To reduce implementation overhead and make use of Linux device drivers, LyraOS provides the Linux device driver emulation environment [26]. Under this environment, Linux device driver codes can be integrated into LyraOS without modification. Therefore, a thread is created for running this network device driver emulation environment. Detail about the implementation of this emulation environment can be referred to the paper [26].

4. PERFORMANCE EVALUATIONS

This section describes the primitive performance evaluation for this ported communication stacks. To measure the maximum data transferring rate, two PCs are directly connected. Figure 12 and Table 2 show the experimental platform.

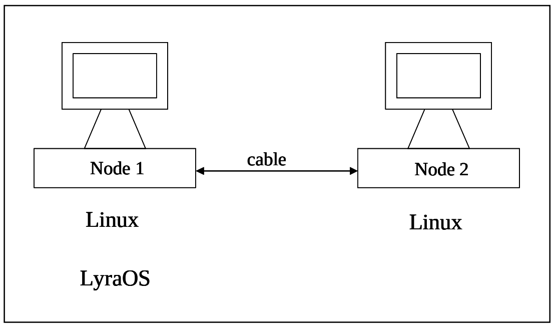

Figure 12: Experimental Platform.

	NODE 1	NODE 2
CPU	Pentium 200 MHz	Pentium 90 MHz
RAM	64MB DRAM	80M DRAM
OS	Linux Kernel version: 2.0.36 and LyraOS	Linux 2.0.35
Network Interface Card	3com 3c509	NE2000

Table 2: Experimental Platform.

4.1 Code Size

This section measures the modification of Linux communication stacks and implementation of socket interface for application use in LyraOS. Table 3 shows the modification and code size about the ported TCP layer. *kern_inf.c* is the added socket interface in LyraOS.

4.2 Data Transfer Rate

This section describes evaluation of LyraOS TCP transmission performance. The maximum data transfer rate is measured. The same evaluation was conducted under Linux and LyraOS in order to compare the performance difference under different systems.

One set of benchmark including client and server programs are created to measure the elapsed time of transferring 10,000 times of 1460-byte packets. Two kinds of evaluation were performed as shown in Figure 13 and 14. One evaluation measured the elapsed time for server to send data; the other measured the elapsed time for server to receive data. Each evaluation was performed 100 times.

Source Codes	Linux Original Lines of Codes	Modified Lines of Codes	Object Code Size
tcp.c	2438	300	15182
tcp_input.c	2771	20	12528
tcp_output.c	1449	20	10699
tcp_timer.c	310	0	2481
kern_inf.c	N/A	336	1447
select.c	300	286	2481

Table 3: Code Size of TCP Layer.

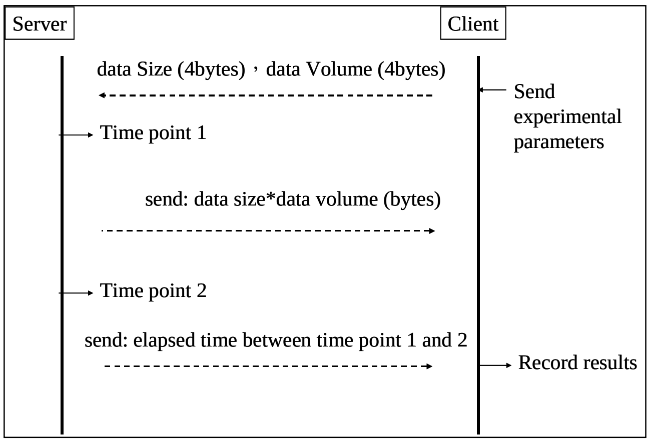
Server
Client
data Size (4bytes)，data Volume (4bytes)
Send experimental parameters
Time point 1
send: data size*data volume (bytes)
Time point 2
send: elapsed time between time point 1 and 2
Record results

Figure 13: Evaluation of Data Send Rate on Server.

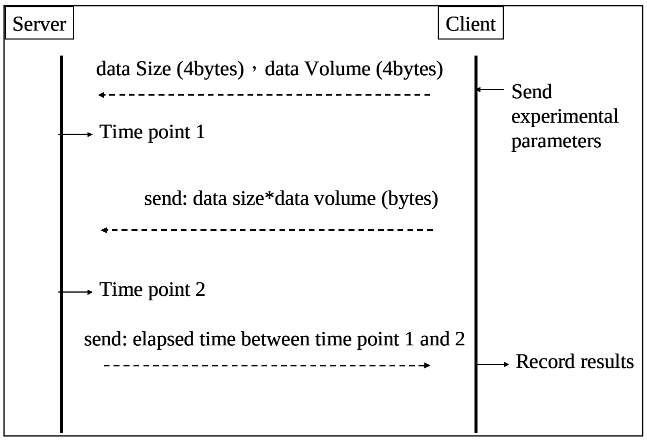
Server
Client
data Size (4bytes)，data Volume (4bytes)
Send experimental parameters
Time point 1
send: data size*data volume (bytes)
Time point 2
send: elapsed time between time point 1 and 2
Record results

Figure 14: Evaluation of Data Receive Rate on Server.

Table 4 shows the evaluation results. In our measurement, LyraOS performed better than Linux by 7.39%. The reason can be explained as follows. First, in LyraOS, no system call invocation is needed for accessing network services, which eliminates the overhead for crossing user mode and kernel mode. Besides, since communication stacks are implemented as a separate component in LyraOS and separated from file system, the overhead to manipulate and access file system data structures are eliminated.

	Average Rate (Bytes per sec.)
LyraOS	1,051,741
Linux	979,285

Table 4: Data Transferring Rate.

5. CONCLUSIONS

To add Internet connectivity to LyraOS, we have successfully integrated Linux communication stacks into LyraOS. In this paper, we have described how to solve the integration difficulties from different system design principles and different kernel architectures. The integration work focuses on two parts as follows. For implementing the communication stacks as a self-contained component, we clarify its import and export interfaces and modify Linux TCP/IP codes to separate them from other Linux kernel functions. Some kernel support modules are implemented in LyraOS for integrating Linux TCP/IP protocols.

Performance evaluation shows that for TCP transmission, LyraOS performed better than Linux by 7.39%. This improvement is because the elimination of overhead from system call invocation and from crossing protection domains. Data copy between user space and kernel space is also removed. Besides, the overhead to access and manipulate file system data structures is also eliminated.

To sum up, the success of this porting and the experience of this integration study can be of practical value to serve as the reference for embedding TCP/IP stacks into target systems which need communication capability.

ACKNOWLEDGEMENT

This research was supported in part by the National Science Council of the Republic of China under grant

No. NSC89-2213-E-001-010 and NSC89-2213-E-260-027.

REFERENCES

[1] Michael Beck, Harald Bohme, Mirko Dziadzka, Ulrich Kunitz, Robert Magnus, and Dirk Verworner, *Linux Kernel Internals*, Addison-Wesley Publishing Company Inc., September 1996.

[2] J. Bruno, J. Brustoloni, E. Grabber, A. Silberschatz, and C. Small, "Pebble: A Component Based Operating System for Embedded Applications," In *Proceedings of 3rd Symposium on Operating Systems Design and Implementation*, USENIX, February 1999.

[3] Zan-Yu Chen, "Draft-LyraOS Component Interface Design Spec. – Machine Dependant Layer, Ver. 1.2," *Technical Report*, Department of Information and Computer Science, National Chiao-Tung University, 2000.

[4] Zan-Yu Chen, "A Component Based Embedded Operating System," *Master Thesis*, Department of Information and Computer Science, National Chiao-Tung University, June 2000.

[5] Zan-Yu Chen, Mei-Ling Chiang, and Ruei-Chuan Chang, "The LyraOS APIs," *Technical Report*, Department of Information and Computer Science, National Chiao-Tung University, 2000.

[6] Zan-Yu Chen, Mei-Ling Chiang, and Ruei-Chuan Chang, "Putting LyraOS onto ÉlanTM µforCE," *Technical Report*, Department of Information and Computer Science, National Chiao-Tung University, 2000.

[7] Mei-Ling Chiang, "Draft-LyraOS Component Interface Design Spec. - Kernel Core, Ver. 1.7," *Technical Report*, Department of Information Management, National Chi-Nan University, 2000.

[8] D. E. Comer, *Internetworking with TCP/IP, Volume I – Principles, Protocols and Architecture*, 2nd edition, Prentice-Hall International, Inc.

[9] D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP, Volume II – Design, Implementation, and Internals*, 1st edition, Prentice-Hall International, Inc.

[10] Luke Deller and Gernot Heiser, "Linking Programs in a Single Address Space," In *Proceedings of 3rd Symposium on Operating Systems Design and Implementation*, USENIX, February 1999.

[11] Embedded Configurable Operating System (eCos), http://www.redhat.com/products/ecos/, 2000.

[12] EPOC, Psion, Nokia, Ericsson, and Motorola, http://www.tcm.hut.fi/Opinnot/Tik-111.550/1999/Esitelmat/Symbian/report.html, 2000.

[13] B. Ford, M. Hibler, J. Lepreau, R. McGrath, and P. Tullmann, "Interface and Execution Models in Fluke Kernel," In *Proceedings of 3rd Symposium on Operating Systems Design and Implementation*, USENIX, February 1999.

[14] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, and O. Shivers. "The Flux OSKit: A Substrate for OS and Language Research," In *Proc. Of the 16th ACM Symp. On Operating System Principles*, Oct. 1997.

[15] GNU General Public License (GPL), http://www.linux.org/info/gnu.html.

[16] GEOS, Geoworks Corporation,

http://www.geoworks.co.uk/os/wireless_big.html.

[17] E. Grabber, C. Small, J. Bruno, J. Brustoloni, and A. Silberschatz, "The Pebble Component-Based Operating System," In *1999 USENIX Annual Technical Conference*, June 1999.

[18] Wen-Shu Huang and R. C. Chang, "An Implementation of a Configurable Window System on LyraOS," *Master Thesis*, Department of Computer and Information Science, National Chiao Tung University, 2000.

[19] Chi-Wei Yang, C. H. Lee, and R. C. Chang, "*Lyra:* A System Framework in Supporting Multimedia Applications," *IEEE International Conference on Multimedia Computing and Systems'99 (ICMCS'99)* Florence, Italy, June 1999.

[20] X. Liu, C. kreitz, R. van Renesse, J. Hickey, M. Hayden, K. Birman, and R. Constable, "Building reliable, high-performance communication systems from components," In 17th *ACM Symposium on Operating Systems Principles*(SOSP' 99), Dec 1999.

[21] MicroC/OS II homepage at http://www.ucos-ii.com/, 2000.

[22] PalmOS homepage at http://www.palmos.com/, 2000.

[23] QNX homepage at http://www.qnx.com/, 2000.

[24] W. R. Stevens, *TCP/IP Illustrated: The Protocols* Volume 1. Reading, MA: Addison-Wesley.

[25] Windows CE homepage at http://www.microsoft.com/embedded/, 2000.

[26] Chi-Wei Yang, Paul C. H. Lee, and R. C. Chang, "Reuse Linux Device Drivers in Embedded Systems," *Proceeding of the 1998 International Computer Symposium*(ICS'98), Taiwan, 1998.