World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:8, 2010

Application of Formal Methods for Designing a Separation Kernel for Embedded Systems

Kei Kawamorita, Ryouta Kasahara, Yuuki Mochizuki and Kenichiro Noguchi

***Abstract*—**A separation-kernel-based operating system (OS) has been designed for use in secure embedded systems by applying formal methods to the design of the separation-kernel part. The separation kernel is a small OS kernel that provides an abstract distributed environment on a single CPU. The design of the separation kernel was verified using two formal methods, the B method and the Spin model checker. A newly designed semi-formal method, the extended state transition method, was also applied. An OS comprising the separation-kernel part and additional OS services on top of the separation kernel was prototyped on the Intel IA-32 architecture. Developing and testing of a prototype embedded application, a point-of-sale application, on the prototype OS demonstrated that the proposed architecture and the use of formal methods to design its kernel part are effective for achieving a secure embedded system having a high-assurance separation kernel.

***Keywords*—**B method, embedded systems, extended state transition, formal methods, separation kernel, Spin.

I. Introduction

EMBEDDED systems have become ubiquitous, and their functionalities are becoming richer and reaching higher levels. Obtaining high-assurance embedded systems is critical in many environments. The layered approach has been adopted to obtain these high-assurance systems. The lowest layer is an operating system (OS) kernel, which needs to have the highest assurance. Attaining high assurance is not easy as today's OS kernels tend to be large and have rich functionality. A small kernel called the separation kernel, which was proposed by Rushby [1] [2], has attracted attention as a potential high-assurance kernel for embedded systems. The separation kernel provides an abstract distributed environment on a single CPU. Because the separation kernel is small, its secure design, including verification of its correctness, is easier than with traditional kernels. The Common Criteria for Information Technology Security Evaluation [5], which defines the requirements for secure systems, calls for a formal design at its highest evaluation assurance level (EAL 7).

This paper reports the results obtained for the secure design and implementation of a separation-kernel-based OS (tentatively called OS-K), which is intended for use in secure embedded systems. The architecture is proposed for a separation-kernel-based system. The separation-kernel layer provides an abstract distributed environment to partitions. It was designed using two formal methods: the B method and the Spin model checker. Also applied was a newly developed semi-formal method, the extended state transition (EST) method. These methods played complementary roles in the verification of the design. The separation-kernel layer and the additional OS services on top of it were prototyped on the Intel IA-32 architecture. The additional OS services comprised the partition OSs in the client partitions and the OS servers in the server partitions. The OS servers included the file server and device drivers. A sample embedded application, a POS (point of sale) application, was developed on the prototype OS, resulting in a POS system in a simulated environment. The development result demonstrated that the proposed architecture and the use of formal methods to design its kernel part are effective for achieving a secure embedded system having a high-assurance separation kernel.

Kei Kawamorita was a graduate course student at Kanagawa University, Japan. He is now with the Hitachi Software Engineering, Inc., Tokyo, Japan. (e-mail: kei.k.0203@gmail.com).

Ryouta Kasahara and Yuuki Mochizuki are graduate course students at Kanagawa University.

Dr. Kenichiro Noguchi is with the Department of Information and Computer Science, Kanagawa University, 2946 Tsuchiya, Hiratsuka-shi 259-1293, Japan (corresponding author to provide phone: +81-463-59-4111; fax: +81-463-58-9684; e-mail: noguchi@kanagawa-u.ac.jp).

II. Separation-kernel-based Architecture

A. Overview

A separation-kernel abstraction was adopted for the OS kernel to enable achievement of a secure and reliable embedded system. Because an OS kernel designed using this abstraction is small and simple, the possibility of kernel failure is minimized. Moreover, proving the correctness of the kernel is easier. The security of the system can be further improved by dividing an application into multiple processes with different privileges and running them in the different partitions provided by the separation kernel.

The overall architecture for the developed separation-kernel-based system is outlined in Fig. 1. The separation-kernel layer provides multiple partitions on top of it. Client-server-mode operations are used to enable the multiple partitions to work together. The client partitions are for the user processes. One user process runs in one client partition. The partition OS in each client partition acts as an interface for providing OS services to the user process. The partition OSs send requests to the server partitions for required services. The server partitions provide common OS services to the client partitions and to other server partitions. Such OS functions as the file server and device drivers, which are located in the OS kernel in traditional OSs (and even in microkernels for device drivers), run in the server partitions as server processes in user (non-privilege) mode, which reduces the possibility of kernel

failure.

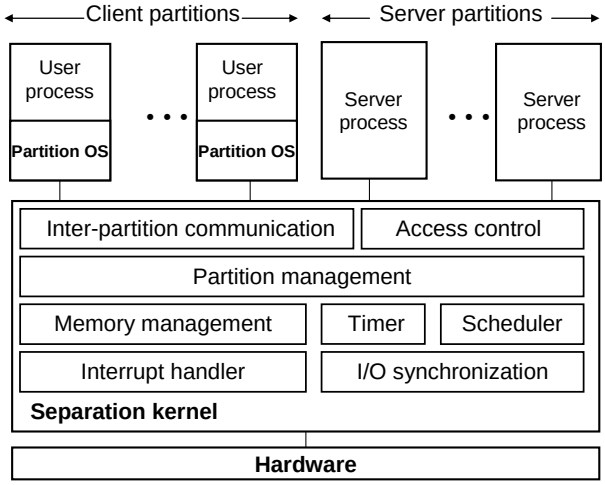


Fig. 1 Architecture of separation-kernel-based system

The separation-kernel layer provides several functions:
- partition management;
- inter-partition communication;
- access control for inter-partition communication;
- memory management;
- timer management;
- processor scheduling;
- I/O interrupt synchronization for device driver operation;
- interrupt-handling.

B. Separation-kernel calls

The separation-kernel layer provides services to the upper layer via separation-kernel calls. The separation-kernel calls are classified into three service categories.

- *Message-passing service*: The main service provided by the separation-kernel layer to the upper layer is inter-partition communication. The client-server-mode message-passing service is used to align the service with the overall architecture of the partition layer. There are four separation-kernel calls for this service: *send*, *receive*, *fetch*, and *reply*. The client side uses *send*, and the server side uses *receive* and *reply*, or *receive*, *fetch*, and *reply*, as a combination. A *send* is issued to transmit a message to the destination partition and wait for a reply message. A *receive* waits for a message that another partition is sending. If a message has already been sent to this partition, a *receive* returns immediately. When a *receive* returns with information that a variable-length message has been sent, a *fetch* is issued to retrieve the message. A *reply* is issued to send a reply message to the partition that issued a *send*, and the partition's wait state is released.
- *I/O synchronization service*: This service is for the device drivers in the server partitions. A device driver issues a *dwait* separation-kernel call after initiating an I/O operation and waits for an I/O completion interrupt.
- *Timer service*: A partition that has issued *sleep* is put into wait state until a specified timer-interval expires. The *gettime* returns the current clock time.

C. Inter-partition access control

The separation kernel provides the access-control function for inter-partition communication, which provides the only linkage between otherwise separate partitions. Therefore, it is critical to maintaining the security of the system. For example, even if the control of a partition is taken over by a malicious program, properly set access control can minimize its effect on other partitions. The access-control function regulates which partition can communicate with which other partitions. The rules for access control are set by a system administrator.

Example access-control rules for inter-partition communication are shown in Table I. Client A is allowed to communicate, i.e., to send a message and receive a reply, with Client B and Server B. Client B is allowed to communicate with Servers A and B. Server A is allowed to communicate with Server C. The separation kernel prohibits other combinations of inter-partition communication.

Running components such as device drivers in the server partitions enables accesses to such components to be controlled by using the access-control function for inter-partition communication.

D. Memory Protection

TABLE I
EXAMPLE ACCESS-CONTROL RULES FOR INTER-PARTITION COMMUNICATION

Source	Destination				
	Client A	Client B	Server A	Server B	Server C
Client A	P	A	P	A	P
Client B	P	P	A	A	P
Server A	P	P	P	P	A
Server B	P	P	P	P	P
Server C	P	P	P	P	P

A: Allowed, P: Prohibited

Effective memory protection is critical to isolate the memory spaces of partitions, which is the key security feature of the separation kernel. There are three requirements for memory protection.

- The memory space of each partition must be isolated from that of the other partitions; i.e., a process in one partition cannot access the memory space of another partition.
- The memory area of the separation kernel must not be accessible by the user processes, the partition OSs in the client partitions, or the processes in the server partitions.
- The memory area of a partition OS must not be accessible by user processes.

In this IA-32-architecture-based implementation of the separation kernel and the upper-layer OS services, two memory-protection features of the IA-32 architecture are utilized.

- The ring protection feature of the IA-32 architecture is used to protect the memory area of the separation kernel against access by the processes and the partition OSs. As illustrated in Fig. 2, the memory area of the separation kernel is assigned privilege level 0, which is the highest level in the system. The memory spaces for the server processes are assigned level 1. The memory areas for the

partition OSs are assigned level 2. The memory spaces for the user processes are assigned level 3.

- Each partition is assigned a local descriptor table in which the partition segments are registered to isolate the partition memory spaces. This prevents programs in one partition from accessing the memory space of another partition although their spaces have identical privilege levels.

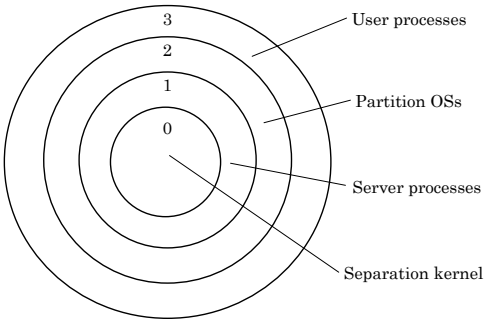


Fig. 2 Assignment of privilege levels in ring protection

III. Formal Design with B Method

As mentioned above, one of the requirements defined by the Common Criteria for Information Technology Security Evaluation [5] for secure systems is formal design at the highest evaluation assurance level, EAL 7. The kernel part that runs in kernel (privileged) mode and controls the whole system needs to have the highest assurance level.

Several formal methods were considered for designing the separation kernel, including Z language [9], the B method [10], [11], and the Spin model checker [14], [15]. The B method is particularly attractive because the abstract description of the specifications can be refined to a more concrete description, i.e., the description of the IMPLEMENTATION in B terminology, and the correctness of the refinement can be verified using the tool associated with the B method. Therefore, the B method was selected for designing the overall structure of the separation kernel.

A. Modeling

The specifications of the separation kernel in the proposed architecture are described in B. The main components in the description and their relationships are outlined in Fig. 3. The components are called abstract machines and are described as MODELs. The *memorymanager* MODEL describes the main function of memory management. The *memorymanager* operations internally call operations in the *msegaccess* MODEL. The *skinterface* MODEL describes the main functions of message passing, scheduler, the timer, I/O synchronization, and the interrupt handler. Because MODEL description in B does not allow sequential processing, the *skinterface* MODEL is necessarily a large component with a nondeterministic description. It was thus refined into the *skinterface_i* IMPLEMENTATION in which the operations internally call operations in the *midmanager*, *mdata_mgr*, *pidmanager*, *sktimer*, *pscheduler*, and *msgpass* MODELs.

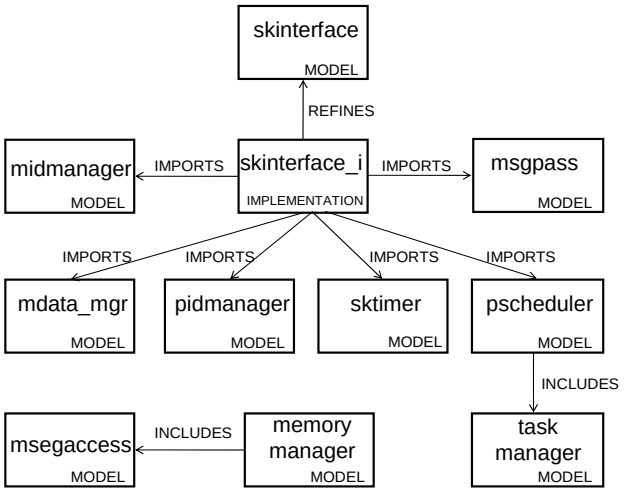


Fig. 3 Main components of separation kernel in B description

An example MODEL description, that of *memalc* operation in the *memorymanager* MODEL, is shown in Fig. 4.

```
ridx <-- memalc(isize) =
  PRE isize : NAT1 & isize mod 4 = 0
  THEN
    CHOICE
      ANY
        idx
      WHERE
        idx : MLIDX & mstate(idx)=m_free & msize(idx) >=
isize
      THEN
        IF msize(idx) = isize
        THEN
          update_state(idx, m_allocd)
        ELSE
          divide_mseg(idx, isize)
        END ||
        ridx := idx
      END
    OR
      ridx := max_segs
    END
  END
```

Fig. 4 Example MODEL description in B: *memalc* operation

B. Verification of Abstract Machines

The correctness, i.e., consistency, of the descriptions was verified using B4free [12]. B4free is a tool for the B method that supports the generation of proof obligations and their proofs. It verifies the proof obligations automatically as much as possible; any that it cannot verify are left for verification by hand. In automatic verification, the tool verifies whether the invariant conditions described in the INVARIANT clause of the operation description hold after the operation has been executed as well as at initialization. It has an interactive verification mode that supports verification by hand.

As shown in Table II, all the proof obligations generated by the tool, except 1 for *memorymanager* and 125 for *msegaccess*, were verified automatically by B4free. The 126 others were verified interactively or manually. The reason for the large number of *msegaccess* ones requiring interactive or manual verification may be that the conditions described in its INVARIANT clause were stricter compared with the descriptions of the other components.

C. Description Refinement

TABLE II
GENERATED PROOF OBLIGATIONS (POS) AND VERIFICATION RESULTS

Component	POs generated	POs proved automatically	POs proved interactively or manually
memorymanager	14	13	1
msegaccess	228	103	125
skinterface	6	6	0
mdata_mgr	2	2	0
midmanager	6	6	0
msgpass	95	95	0
pidmanager	6	6	0
pscheduler	62	62	0
sktimer	180	180	0
taskmanager	1	1	0

The abstract descriptions of the specifications were refined to the IMPLEMENTATION descriptions by converting the nondeterministic sections to sequential processing. An example refined description is shown in Fig. 5. It is for the *memalloc* operation and corresponds to the abstract description in Fig. 4.

```
ridx <-- memalc(isize) =
  BEGIN
    VAR idx IN
      idx <-- search_fmsegbysize(isize);
      IF idx < max_segs
      THEN
        IF msize(idx) = isize
        THEN
          update_state(idx, m_allocd)
        ELSE
          divide_mseg(idx, isize)
        END;
        ridx := idx
      ELSE
        ridx := max_segs
      END
    END
  END
```

Fig. 5 Refined description in B: *memalc* operation

D. Verification of Refinement

The consistency of the refined descriptions was verified using B4free. Both the consistency of the description itself and the consistency between the abstract description and the refined description were verified.

As shown in Table III, all the proof obligations generated by the tool for components other than *memorymanager_i*, *msegaccess_i*, and *skinterrface_i* were automatically verified. A small number of them for *memorymanager_i* and *msegaccess_i* were verified interactively or manually. The number of non-trivial proof obligations exceeded 1,000 for *skinterface*, which resulted in an error and could not be verified.

TABLE III
GENERATED PROOF OBLIGATIONS (POS) AND VERIFICATION RESULTS

Component	POs generated	POs proved automatically	POs proved interactively or manually
memorymanager_i	214	213	1
msegaccess_i	244	220	24
skinterface_i*	Over 1000	0	0
mdata_mgr_i	15	15	0
midmanager_i	20	20	0
msgpass_i	104	104	0
pidmanager_i	20	20	0
pscheduler_i	56	56	0
sktimer_i	354	354	0

*Verification for skinterface_i is incomplete.

IV. MODEL CHECKING WITH EXTENDED STATE TRANSITION METHOD

The extended state transition (EST) method was applied in parallel with the B method. The EST model extends the normal state-transition model by incorporating variables into behavior descriptions. Its application to software design is described in [13]. A scheme was developed to describe the model of the specifications on the basis of the EST model, and tools were developed to check the validity of the model description by executing it. Since it does not yet have exhaustive model verification capability, it is a semi-formal method.

A. Modeling

Selected functionalities of the separation-kernel specifications, i.e., inter-partition communication, scheduler and timer are included in the model. In the model description (see Fig. 6), the *partition entities*, the *scheduler*, and the *timer* are modeled as extended state machines. A *partition entity* controls the operation of each partition. The *upper partition* is the non-kernel part of a partition; it uses the separation kernel by issuing separation-kernel calls.

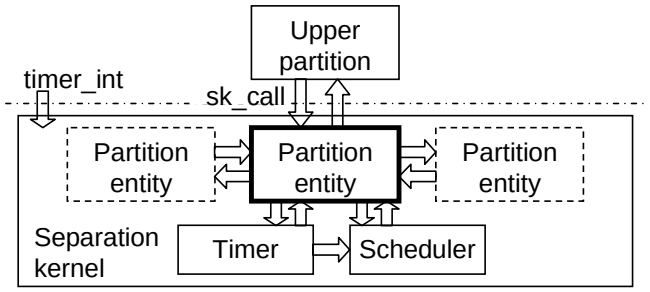


Fig. 6 Configuration of model description of separation kernel

The *partition entity* model has seven major states, which do not include finer states represented by variables. The

state-transition diagram for the major states is shown in Fig. 7. When a *partition entity* receives a *send* kernel call, it internally issues an *isend* (internal send) request to the destination's *partition entity*. Similarly, when a *reply* kernel call is received, an *ireply* (internal reply) request is internally issued.

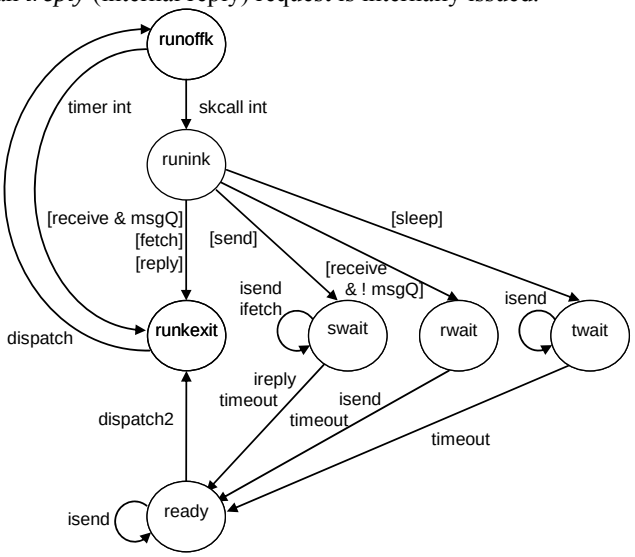


Fig. 7 State transition diagram of partition entity (for major states)

The model of the specifications is described in extended-state-transition diagrams, i.e., by allowing the incorporation of variables into behavior descriptions associated with the state transition. To enable descriptions to be analyzed by programs, we formulated the XML format for extended-state-transition diagrams. Fig. 8 shows a part of description of partition entity specification.

```
<statemachine>
   :
  <input name = "SEND">
    <predicate name = "PREDICATE1">
       :
    </predicate>
    <predicate name = "PREDICATE2">
       Condition = sk.skmode;
      <state name = "RUNINK">
        <action>
           if (in.val == pid || ! existPID(in.val)) {
              ret = returnInfo(ERROR);
              state = RUNKEXIT;
              Output(SCHEDULER, SCHEDULE);
           } else {
              timeout = in.val2;
              in.val2 = genID();
              Output(in.val, ISEND, in.val, in.val2, in.str);
              saveSendInfo(in.val, in.val2);
           }
        </action>
      </state>
      <stategroup name = "default">
        <action dontcare = "true">
        </action>
      </stategroup>
    </predicate>
  </input>
  <input name = "RECEIVE">
     :
```

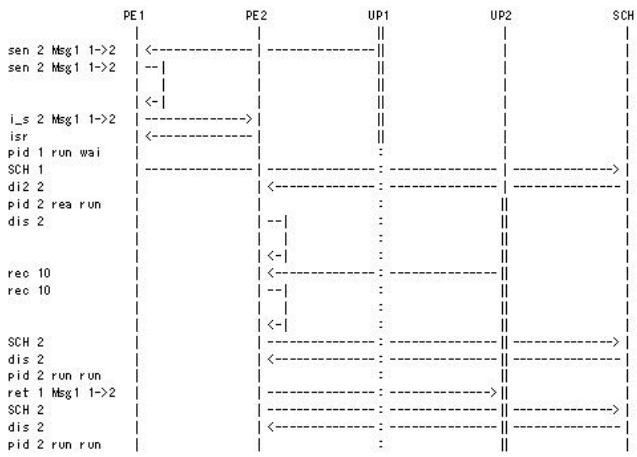
Fig. 8 Part of description of partition entity specification (XML format)

B. Model Checking

The EST method has a tool, the *converter*, that converts the description in XML format into a Java method. Another tool, the *specification-execution-environment*, receives an execution scenario as input, executes the Java methods, and outputs the execution results. An execution scenario is composed of the descriptions of the upper partitions. The results consist of log information, sequence diagrams generated from the log information, coverage information on the state transitions, and detailed information on the system's state, including the values of the variables at the beginning and the end of the execution. Fig. 9 shows a part of the automatically converted Java method from the description in Fig. 8. Fig. 10 shows an example of a generated sequence diagram.

```
public boolean StateMachine(IOObj in) {
   :
  switch (in.type) {
   case SEND:
     :
     Condition = sk.skmode;
     if (Condition) {
       switch (state) {
         case RUNINK:
           << Content of action here>>
           break;
         default:
           systemError("Input not acceptable in this state");
           return false;
       }
       break;
     }
   case RECEIVE:
     :
```

Fig. 9 Part of generated Java method for partition entity



|| running | ready : waiting

Fig. 10 Example generated sequence diagram (partial view)

The model of the specifications was executed under various scenarios, and the validity of the results was checked. It was found that the EST method could be effectively used for testing with tentative specifications by executing the model and then revising it on the basis of the results. The method was particularly useful for finalizing the specifications for inter-partition communication.

V. Model Checking with Spin

The Spin model checker is primarily aimed at the design of concurrent systems. Because a separation kernel provides an abstract distributed environment, the Spin model checker is potentially an effective tool in its design. It was thus used for the design described here.

A. Modeling

The Spin model checker was applied to the same scope of separation-kernel functionalities as the EST method. The model description had the same composition as the EST model outlined in Fig. 6. As a result, the Spin model could easily be obtained by directly translating the descriptions of an EST model into those in the Spin model description language, Promela.

The entities in Fig. 6 are described as *active proctypes* in Promela:

- an array of *partition entities*
- an *upper partition* for each *partition entity*
- *scheduler*
- *timer*
- *idle partition entity*.

These entities are connected through the Promela input/output channels.

The major states of a *partition entity* and the transitions among them illustrated in Fig. 7 also apply to the Spin model. There is no nondeterminism in the description of the separation kernel.

B. Verification

The checking of the model using Spin was done in two modes, simulation and verification. The model was checked in simulation mode with various descriptions of the upper partitions. Random simulations, done by incorporating nondeterminism into the descriptions of upper partitions, enabled the efficiency of the checking to be enhanced. Some errors in the Promela model coding were found.

With nondeterministic descriptions in the upper layers, an exhaustive check of the model was performed in verification mode. The correctness property of successful message passings was verified using assertions. The safety property of the model was verified by checking that the upper partitions were always served to the completion by the separation kernel and that the system reached the end state. The model checking in the verification mode was possible for relatively small models with at most three *partition entities*, due to CPU time and memory space constraints in the verification run and the size of the model descriptions (about 1,000 lines in Promela).

Fig. 11 shows an example nondeterministic description of an upper partition. Inline constructs were used to code the separation-kernel calls.

Because the model checking with Spin was carried out after that with the EST method, no feedback was obtained for the specifications of the separation kernel. Better confidence in the validity of the specifications, however, was gained.

```
active proctype UP1()
   provided (! skmode && current == _pid - PENUM)
{
   :
   :
  do
  :: count1 > 0 ->
     if
     :: sendf(dest, 10, val1, result, from, msgid, val2)
     :: receivef(20, result, source, msgid, val2)
        val1 = val2 + 1
     :: replyf(source, msgid, val1, result)
     :: yieldf()
     fi;
     count1--
  :: count1 == 0 -> break
  od;
  exitf()
}
```

Fig. 11 Example nondeterministic description of upper partition

VI. Prototyping

The separation kernel and other OS components were implemented on an IA-32-architecture [16] processor after the application of the B and EST methods to the design of the separation kernel.

A. Separation-kernel Layer

The separation kernel was implemented on an IA-32-architecture processor starting with a boot program. The IMPLEMENTATIONs of the separation kernel in B were converted into C programs on an almost one-to-one basis. The program implemented for the separation kernel had about 3,000 lines of C code and about 1,000 lines of assembler code.

B. Partition OSs

The partition OSs that provide system calls to user processes to access the following services were prototyped.

- *File-management service*: This service has five system calls commonly found in file systems: *open*, *read*, *write*, *seek*, and *close*.
- *Standard I/O service*: This service has two system calls: *prints* and *scans*. A *prints* is issued to display an output string. A *scans* is issued to retrieve an input string from the keyboard.
- *Inter-client-partition communication*: This service has three system calls: *put_request*, *get_request*, and *done_request*. A *put_request* is issued to send a message to another client partition. A *get_request* is issued to receive a message from a client partition. A *done_request* is issued to reply to a message sent to the client partition.

The partition OSs call the server partitions to fulfill the file-management and standard I/O services.

C. OS Server Layer

Three server partitions (the disk driver, the terminal driver, and the file server) were implemented.

(1) Disk driver

The disk driver provides two functions to the file server, *disk_read* and *disk_write*. A *disk_read* is issued to read data on the disk. A *disk_write* is issued to write data to the disk. The ten I/O ports required for the disk driver to work were made available.

(2) Terminal driver

The terminal driver provides standard I/O service to the other partitions. The service is called by the standard I/O service in the partition OSs. The two I/O ports required for the terminal driver to work were made available.

(3) File server

The file server provides a file-management service to the other partitions. This server is called by the file-management service in the partition OSs. The format of the file in the file server is FAT16. The file server calls the disk driver to read data from and write data to the disk.

D. Access Control

Access-control services were implemented in the upper layer. These services are independent of the inter-partition access control in the separation-kernel layer and are intended to provide the finer access control needed for building secure and reliable embedded systems.

(1) I/O port access

The I/O ports available to the partitions must be controlled to minimize the effect if a partition is attacked. I/O-port-access privileges were assigned to the components in accordance with the privilege levels of the IA-32 architecture, as summarized in Table IV. I/O-port-access privileges are not granted to user processes and partition OSs. Instead, privileges are granted to server partitions on the basis of their type. The separation kernel can access all I/O ports.

TABLE IV
DEVICE I/O PRIVILEGES

Privilege level	I/O port privileges
3	None
2	None
1	Granted on basis of server type
0	All

(2) File access

An access-control function in the file server was implemented using the Bell-LaPadula model [17] as the access-control model for controlling client partitions that can communicate with the file server and access files. Client partitions and files are assigned confidential levels. Information at a higher level is thus prevented from leaking to lower ones.

E. Sample Application

A POS application was prototyped on the prototyped OS. As illustrated in Fig. 12, a POS register receives a merchandise code, a customer code and received money amount as input. The POS register manages payment, merchandise, sales, and customer services. It also manages customer personal data, customer's reward point data, merchandise data, and sales history data.

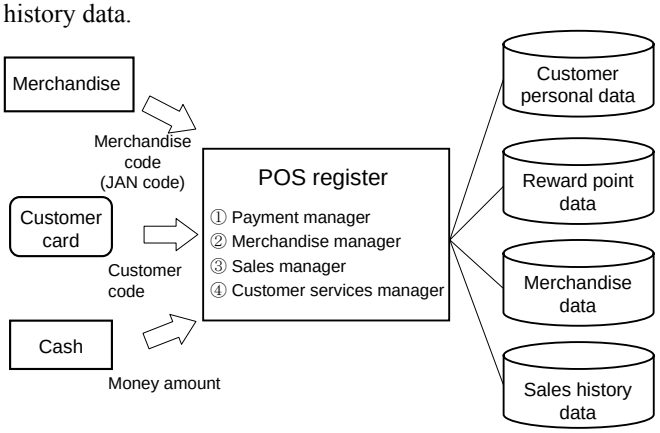


Fig. 12 Overview of sample application

The services for users are divided into ten client partitions: function selecting, payment, customer-information reference, etc.

The access-control rules for inter-partition communication are listed in Table I: Clients A and B correspond to function selecting and the other client partitions; Servers A, B, and C correspond to the file server, the terminal driver, and the disk driver, respectively. Function selecting is allowed to communicate with the other client partitions corresponding to functions required by the user and with the terminal driver to receive input data from the keyboard and to output messages to the user. The other client partitions are allowed to communicate with the file server and the terminal driver. The file server is allowed to communicate with the disk driver. The separation kernel prohibits other combinations of inter-partition communication.

The confidential levels are assigned to client partitions and files for the access control provided by the file server. Customer-information reference is assigned the highest level among the partitions, and customer personal data is assigned the highest level among the files.

VII. DISCUSSION

A. Separation-kernel-based architecture

The advantages of the proposed separation-kernel-based architecture for secure embedded systems were verified and demonstrated through the design and development of the kernel and an application in a sample embedded-system environment.

Four advantages in particular contribute to achieving secure embedded systems.

- The separation kernel can be kept small, minimizing the security-critical kernel-mode program. Such OS services as a file server and device drivers can be implemented as non-kernel-mode programs. Even if a file server or device driver is taken over and controlled by a malicious program, it cannot control the kernel.
- A small kernel makes it easy for the correctness of its design to be formally verified.
- A separation kernel provides an isolated environment for

multiple components of an embedded application, enabling a malfunction in or a security attack against some component to be localized in that component, thereby minimizing its effect on other components. The segregation of partition memory spaces and of the separation kernel and partition memory areas can be effectively achieved in the IA-32-hardware environment by making use of ring-protection and segment mechanisms.

- The access-control function for inter-partition communication controls the communication channels between partitions and prevents unwanted flows of information between partitions.

Because the separation kernel is small and its functionality is limited, it alone cannot provide all the OS services needed by embedded applications. The remaining services need to be provided as upper-layer OS services. The design and development of such upper-layer OS services reported here have demonstrated that they can be designed to be secure by taking advantage of the separated partitions provided by the separation kernel and that appropriate security functions need to be provided by them to complement the security functions provided by the separation kernel. The upper-layer OS services offer advantages and additional functionalities.

- Additional OS services are implemented as distributed services. Each service is in its own partition, isolated from other services, which improves the security and reliability of the system. Additionally, access control to such services is established by making use of the access-control function for inter-partition communication provided by the separation kernel. A service can be provided to applications and other services on the basis of the least-privilege policy.
- Additional access-control functions are implemented in the upper-layer OS services. The file-access-control function is implemented in the file server. The prototype application makes use of this, restricting access to confidential information such as customer data to limited applications on the basis of the least-privilege policy. Access control to I/O ports is implemented in the device drivers.

B. Application of formal methods to kernel development

The B method was used in designing and implementing the separation-kernel part of the proposed architecture. Experience has shown that the effectiveness of its application to secure kernel design depends on the components.

- The B method is effective for such components as those for memory management, where verification of the static properties of the model is a key part of the design process. Memory addresses can be represented mathematically as a set, and memory properties can be precisely described.
- Its effectiveness is limited for such components as those for inter-partition communication, the scheduler, and the timer, where sequential, i.e., dynamic, properties are important. This is because an abstract machine cannot verify dynamic properties. For inter-partition communication and the timer, much of the static properties description was nondeterministic, so complete verification was not possible. For the scheduler, there was not much description of the static properties.

The EST method, which is based on the extended state-transition model, was used to check the model.

- It was used to check inter-partition communication, the scheduler, and the timer. It revealed several design flaws that had a sequential nature, indicating that the EST method is effective for checking the dynamic properties of the model.
- Since it does not work well for mathematical descriptions of the properties of variables, is not effective for checking static properties. Since memory management has properties that are mostly static, it was excluded from the checking.
- The functionality of this method is limited compared with Spin; e.g., it lacks an exhaustive verification feature. It is effective for building a well-structured model based on the EST model.

Spin was applied after the model was checked with the EST method.

- Spin was effective for verifying the dynamic properties of the model. In simulation mode, it provided the same level of model checking as the EST method. Further, its nondeterministic capability was effective in improving the efficiency of the checking.
- Spin has a powerful verification feature, so model checking in verification mode resulted in better confidence in the validity of the separation-kernel specifications. Due to CPU time and memory space constraints, it was applied to relatively small models.

Following application of the B and EST methods to the design, the actual coding of the separation kernel was undertaken. Very few bugs (less than ten) have since been found in the separation-kernel code.

VIII. Related Work

Secure OSs, such as SELinux and LIDS for Linux, have enhanced security features such as access control with fine granularity. SELinux has been applied to embedded systems [6]. Although stringent access control was provided to applications, OS-server programs, such as file systems and device drivers, were not the target of control. Therefore, if their control is taken over by a malicious program, its effect can spread throughout the entire kernel.

The Least Privilege Separation Kernel (LPSK) [3] provides access control with finer granularity than that with the traditional separation kernel. In the traditional separation kernel, the subject of access control is in the partition. However, the partition in LPSK consists of one or more subjects and resources. The access-control policy is defined in subjects and resources. LPSK more stringently provides the principle of least privilege with this approach.

The specifications of a separation kernel were described with TAME (Timed Automata Modeling Environment) and verified

with a prover called the Prototype Verification System [4].

Application of Z notation to kernel development has been reported in [7] and [8], including the verification of a separation kernel [8]. Although Z has advantages due to description and verification at the top level, verification of the correctness of a refinement is not easy.

Application of Spin to process scheduling in a distributed operating system has been reported in [15].

IX. Conclusion

A separation-kernel-based OS has been designed for use in secure embedded systems by applying formal methods to the separation-kernel part. The architecture was proposed for a separation-kernel-based system. The design of the separation-kernel part was verified using two formal methods, the B method and the Spin model checker. A newly developed semi-formal method, the extended state transition (EST) method, was also applied. The effectiveness of these three methods for kernel design was complementary, and applying them was effective in attaining a well designed separation kernel.

The separation-kernel part and additional OS services on top of the separation kernel were prototyped on the Intel IA-32 architecture. The additional OS services were designed on the basis of the client-server model and comprised partition OSs in the client partitions and OS servers in the server partitions. A sample embedded application, a POS application, was also developed on the prototype OS, resulting in a POS system in a simulated environment. This application took advantage of the abstract distributed environment provided by the prototype OS.

The results of this work demonstrate the feasibility of the proposed architecture for designing secure embedded systems. They also show that design using formal methods can be used to effectively create a secure and reliable kernel.

References

[1] J. Rushby, "The design and verification of secure systems," in *ACM Operating Systems Review*, vol. 15, no. 5, *Eighth ACM Symposium on Operating System Principles (SOSP)*, 1981, pp. 12–21.

[2] J. Rushby, "Proof of Separability—A verification technique for a class of security kernels," in *Proc. 5th International Symposium on Programming*, vol. 137 of *Lecture Notes in Computer Science*, 1982, pp. 352–367.

[3] T. E. Levin, C. E. Irvine, and T. D. Nguyen, "Least privilege in separation kernels," in *Proc. International Conf. on Security and Cryptography SECRYPT 2006*, 2006, pp 355–362.

[4] C. L. Heitmeyer, M. Archer, E. I. Leonard, and J. D. McLean, "Formal specification and verification of data separation in a separation kernel for an embedded system," in *ACM Conf. on Computer and Communications Security*, 2006, pp. 346–355.

[5] *Common Criteria for Information Technology Security Evaluation*, Version 3.1 Revision 2, CCMB-2007-09-003. Common Criteria Project Sponsoring Organizations, 2007.

[6] Y. Nakamura and Y. Sameshima, "SELinux for Consumer Electronics Devices," in *Proc. the Linux Symposium*, vol. 2, 2008, pp. 125–134.

[7] I. D. Craig, *Formal Design for Operating System Kernels*. London: Springer-Verlag, 2006.

[8] I. D. Craig, *Formal Refinement for Operating System Kernels*. London: Springer-Verlag, 2007.

[9] B. Potter, J. Sinclair, and D. Till, *An Introduction to Formal Specification and Z*. Prentice Hall, 1991.

[10] J.-R. Abrial, *The B-Book —Assigning programs to meanings*. Cambridge University Press, 1996.

[11] *B Language - Reference Manual*. ClearSy. Available: http://www.b4free.com.

[12] *B4free*. ClearSy. Available: http://www.b4free.com.

[13] K. Noguchi, *Logical Method for Software Design*. Kyoritsu Publishing, Japan, 1990. (in Japanese)

[14] G. J. Holzmann, *The Spin Model Checker: Primer and Reference Manual*. Addison Wesley, 2004.

[15] G. J. Holzmann, "The Model Checker Spin," in *IEEE Trans. on Software Engineering*, vol. 23, no. 5, 1997, pp. 1–17.

[16] *IA-32 Intel Architecture Software Developer's Manuals.* Intel Corporation.

[17] D. Bell and L. LaPadula, *Secure Computer System: Mathematical Foundations and Model.* MITRE Rep. MTR 2547, 1973.