LP49: Embedded system OS based on L4 and Plan 9

Yoshihide Sato *1*2
Katsumi Maruyama *1
{satoh-yoshihide, maruyama}@nii.ac.jp

*1 : National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
*2 : Hitachi, Ltd.
1-6-6 Marunouchi, Chiyoda-ku, Tokyo, Japan

1 Introduction

1.1 Distributed processing and embedded systems / home servers

Distributed processing is required in embedded systems and home servers (From here on in this paper, embedded systems will be taken to include the idea of home servers). The distributed processing model of Plan 9 is very effective and flexible, and it is attractive for embedded systems. The 9P protocol is useful for inter-system communication. The private name space of Plan 9 also enables flexible and safe distributed processing in embedded systems.

1.2 Easy program development of embedded systems

Making program development easy is very important. Plan 9 is a smart component-oriented OS: File servers, etc., are organized as normal processes. Kernel devices (e.g. devxxx) are also component-oriented. If the OS functions can be extended merely by adding components, an embedded system developer can always use the optimum OS. In addition, the system servers in Plan 9 are user mode processes; hence, Plan 9 enables easy program development.

Embedded systems have many different device drivers and an easy way of developing them is desired. It is reported that about 70% of Windows-OS kernel bugs are caused by device drivers. Device drivers are hoped to be able to be placed in user-mode processes instead of in kernel programs.

An embedded system must run many threads effectively. This point is also important. Most embedded systems require efficient concurrent processing, and light-weight thread facilities are hoped. Processes of Plan 9 might be too heavy in some applications.

1.3 Robust system

Embedded systems must be robust. It is very bad that a bug in a device driver should cause a system crash. The effect of a bug should be kept within the module. Every module should be as independent as possible from other modules, and even a single module should be able to start up, again after it goes down.

1.4 LP49: Collaboration of L4 and Plan 9

We are implementing a new OS that has the merits of Plan 9 and a micro-kernel. We have adapted the L4 microkernel and are building the Plan 9-based OS modules on L4. The L4 was chosen for the following reasons:

(1) L4 is a small micro-kernel, serving our purposes.
(2) L4 provides very efficient threads.
(3) L4 has a very efficient and flexible IPC function.

A micro-kernel OS has many processes, and they run with intercommunication. Hence, the IPC's performance dominates the performance of the OS. The L4 microkernel's high-speed IPC enables us to build a high-performance OS, and Plan 9's smart facilities attain flexible distributed processing:

a) Robust, Anti-failure OS
b) component-oriented OS
c) Distributed system
d) Effective message passing
e) Multiple thread processing

2 Basic Idea

The assumed applications of LP49 are embedded systems. Such applications use various kinds of devices, and hence, many kind of device drivers must be developed. Device drivers are apt to have bugs. Bugs in kernel-level device drivers often cause the system to crash; therefore, it would be better if device drivers are placed outside the kernel. Hence, LP49 has user-level device drivers, and OS user can change even if only one device driver into new driver. These features make debugging easier.

LP49 will implement the 9P protocol, so it can communicate with Plan 9.

The following outlines LP49:

a) Robust, Anti-failure OS
   The kernel of LP49 is a micro-kernel; hence failures have the minimum effect.
   LP49 saves the running image of the server process, and if the server goes down, LP49 will load this image.
   If a failure happens, LP49 will be able to continue only by restarting the failed servers.
b) component-oriented OS
   LP49 has an OS service as a component, and LP49 supports dynamic plug-in and plug-out.
   It is easy to develop additional functions (including device drivers) for LP49.
   We use Plan 9, which treats devices as components.
   LP49 fits in any embedded system.
c) Distributed system
   Users can transparently use all resources on the network by mounting of remote name spaces of Plan 9.
d) Effective message passing
   LP49 is a micro-kernel-based OS that uses high-speed IPC of the L4 microkernel and maintains high performance.
e) Very light weight thread
   LP49 uses the multiple thread capability of the L4 microkernel.
   LP49 can support many threads for embedded systems.

3 Architecture

Figure 3.1 shows the architecture of LP49.

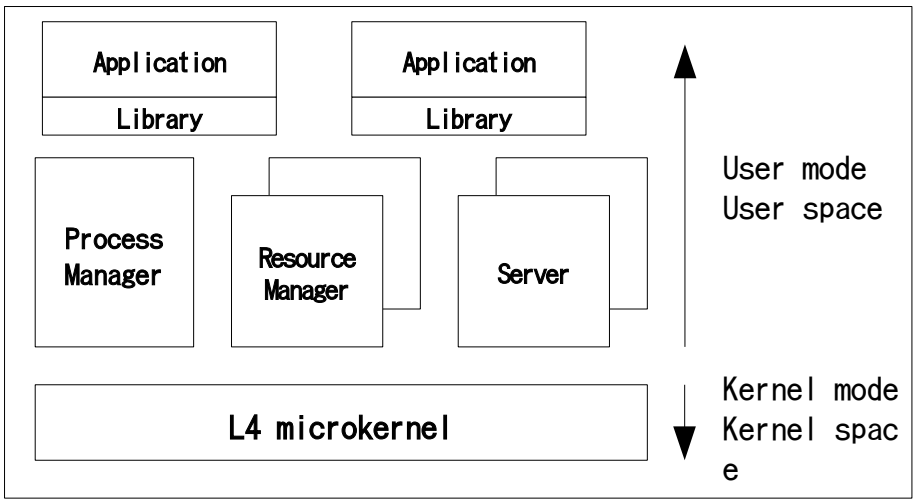

Figure 3.1 LP49 architecture

A "process" is the fundamental and formal unit of program protection and failure isolation. As stated above, device drivers are apt to have bugs, and they are hoped to be put in separate user-level processes to minimize their effect. We planned to separate the device drivers from the Plan 9 kernel, and make the device drivers part of a new user-level process called a "Resource Manager" (RM). Note that if an RM has all the device drivers, the damage that would be caused if it crashed would be very substantial. Hence, the device drivers are distributed in a number of RMs. Consequently, LP49 needs a process that manages other processes. We call this process the "Process Manager" (PM).

・L4 microkernel

The kernel provides logical space, threads, IPC, and page mapping.

・Process Manager

The PM creates and manages processes and threads.
The PM manages name spaces.
In the create() and open() system calls, the PM searches for a suitable RM.

・Resource Managers

RMs are servers that provide the services of a kernel device driver of Plan 9 (i.e., devxxx, resource tree).

RMs can be split into any number of processes.

A process in the same node throws an IPC message to the RM. The RM returns the result to the process.

(RM doesn't speak the 9P protocol. Hence, the RM can't be directly accessed from a remote node.)

・Servers

The Servers of LP49 are almost equivalent to those of Plan 9.

The Servers speak the 9P protocol. If they are exported can be directly accessed from remote nodes.

・Other topics

All processes except for the L4 microkernel are user-mode processes. Of course, device drivers and the interrupt handler are also user-mode processes.

Communication between an application and the PM/RM is done using IPC of L4.

4 How system calls, interrupt handling, etc., work

In LP49, certain user-mode processes provide the functions of the Plan 9 kernel. This section describes the difference between LP49 and Plan 9.

4.1 System calls

In Plan 9, system calls cause a process switch into kernel mode, and requests are serviced inside by kernel. In LP49, system calls cause IPCs to server processes.

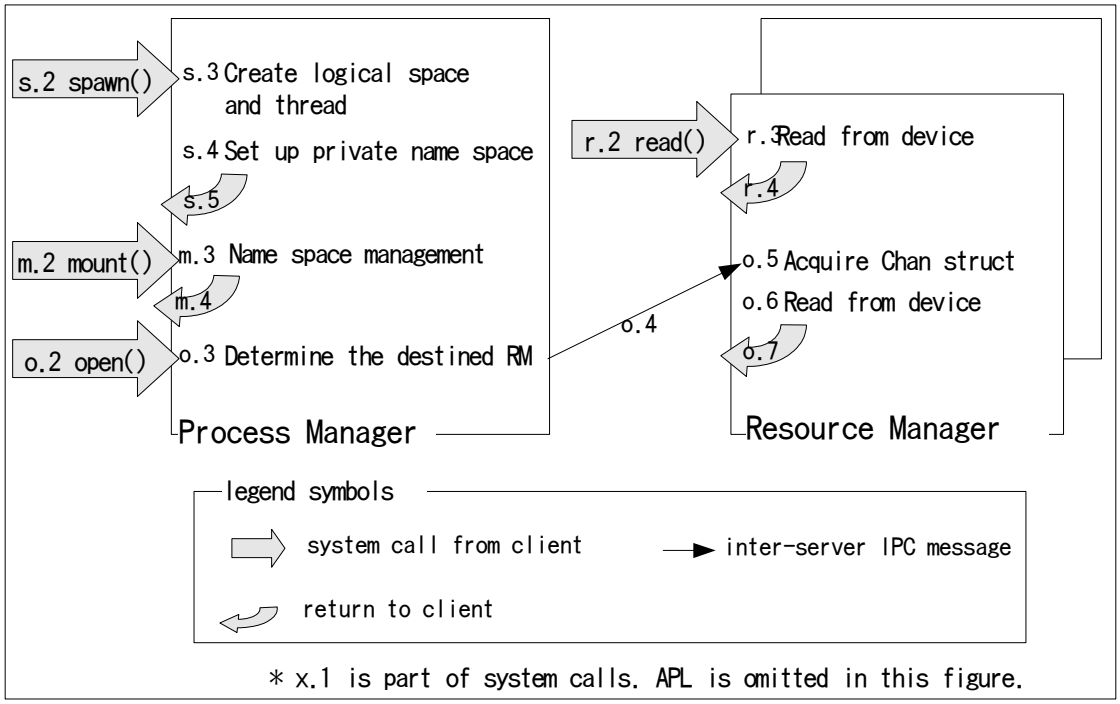

Figure 4.1 Sequence of typical system calls

Figure 4.1 and the following list show how system calls work. The symbols in Figure 4.1 (e.g. m.1, o.1) correspond to the symbols of the following.

(1) spawn()

Instead of fork(), LP49 provides spawn(). spawn() is processed as follows.

s.1 spawn() is called in APL(modified libc).
s.2 A message(L4 IPC) is sent to the PM.
s.3 The PM creates the logical space and thread by invoking L4.
s.4 The name space of the process is set up.
s.5 Reply is returned to the APL.

(2) mount()

m.1 mount() is called in APL.
m.2 Message is sent to the PM.
m.3 The PM modifies the private name space of APL.
m.4 A reply is returned to the APL.

(3) crate() and open()

o.1 create() / open() is called in APL.
o.2 A message is sent to the PM.
o.3 The PM determines the destination RM, which serves the requested resource or file systems, by using the path name parameter and private name space.
o.4 The request is sent to the selected RM.
o.5 The RM creates a new Chan structure for this session.
o.6 The RM creates / opens the requested resource.
o.7 A reply is returned to the APL.

The PM manages process and private name space of any process. Thus, the PM can find the target RM by using the name space and path name parameter.

(4) read(), write(), etc.

r.1 read() / write() is called in APL.
r.2 A message is sent to the RM.

(The library can find the target RM from the file descriptor.)

r.3 The RM reads from / writes to the devices using the Chan structure.
r.4 A reply is returned to the APL.

4.2 Interrupt handling

L4 allows user-level interrupt handling. L4 catches the interrupt and sends an IPC-message to the registered user-level thread.

Figure 4.2 shows the interrupt procedure. A brief explanation is as follows.

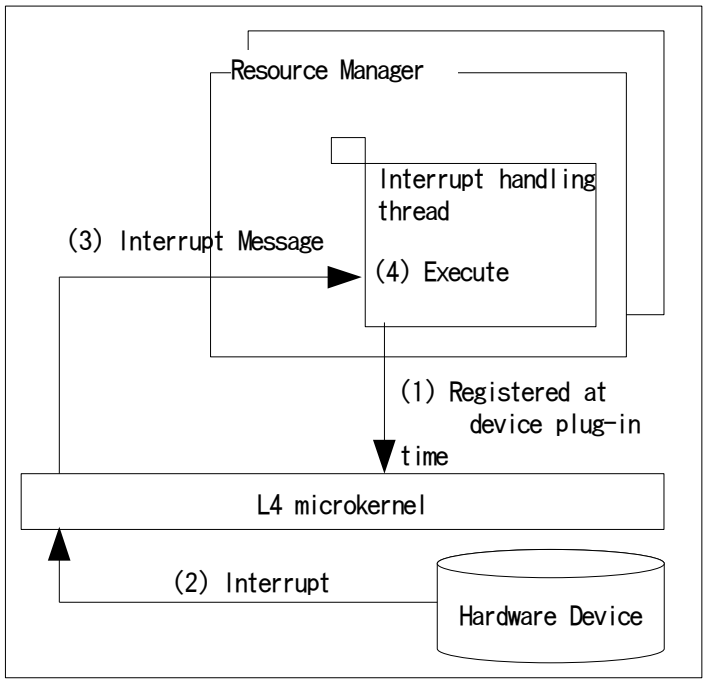

Figure 4.2 Interrupt sequence

(1) When a driver program is loaded, the interrupt handling thread in the RM is registered in the L4 microkernel.
(1) Interrupt occurs.
(2) The L4 microkernel notices the interrupt, and sends the interrupt message to the registered interrupt handling thread.
(3) The interrupt handler is executed.

4.3 9P-based distributed processing

LP49 will support the 9P protocol. Therefore, it will be able to import and mount remote resources, and export local resources to remote node(i.e. import, exportfs).

5 Robust system

5.1 Process replication

The L4 microkernel allows users to provide their own paging algorithm: Pagers are user-defined threads which manage page mapping. When a thread is created, its pager is specified and bound. When a thread accesses an unmapped logical address, a page fault message is sent to the bound pager. The pager, on receiving the page fault message, allocates the requested page.

The pager can access any logical page of any process. In addition, the pager can take a snapshot of the process memory space, and save it for later use. Another usage is for it to save address space before running a procedure, and it can easily do roll-back.

When a server process goes down, LP49 does the following: matters.

(1) It reloads the saved running-image.

(2) It restarts all threads in the process using the same thread ID as when it was running.

The reason for using the same thread ID is that the L4 IPC destination is specified by the thread ID. If the thread ID changes, the IPC from the client can't reach the server process. A server process usually has a number of threads. Hence, the PM must have a thread ID table and be able to search all thread IDs in any server process.

6 Current progress

6.1 Development environment

We chose to use the GCC environment instead of the Plan 9 compiler environment because:

- The L4 microkernel requires the GCC compiler.
- We are accustomed to the GCC environment.

6.2 Plan 9 program porting on L4 + GCC

The language specifications of the Plan 9 compiler are different from those of GCC, and source code modifications are required in the porting.

Automatic conversion is possible, but we are converting them by hand. The most bothersome ones are the "anonymous field" in struct{...}. There are too many anonymous fields. We think anonymous fields are not a good idea, as they degrade program readability.

The automatic linking mechanism of "#pragma lib" is very good, and we hope that GCC will to have this facility.

6.3 Current status

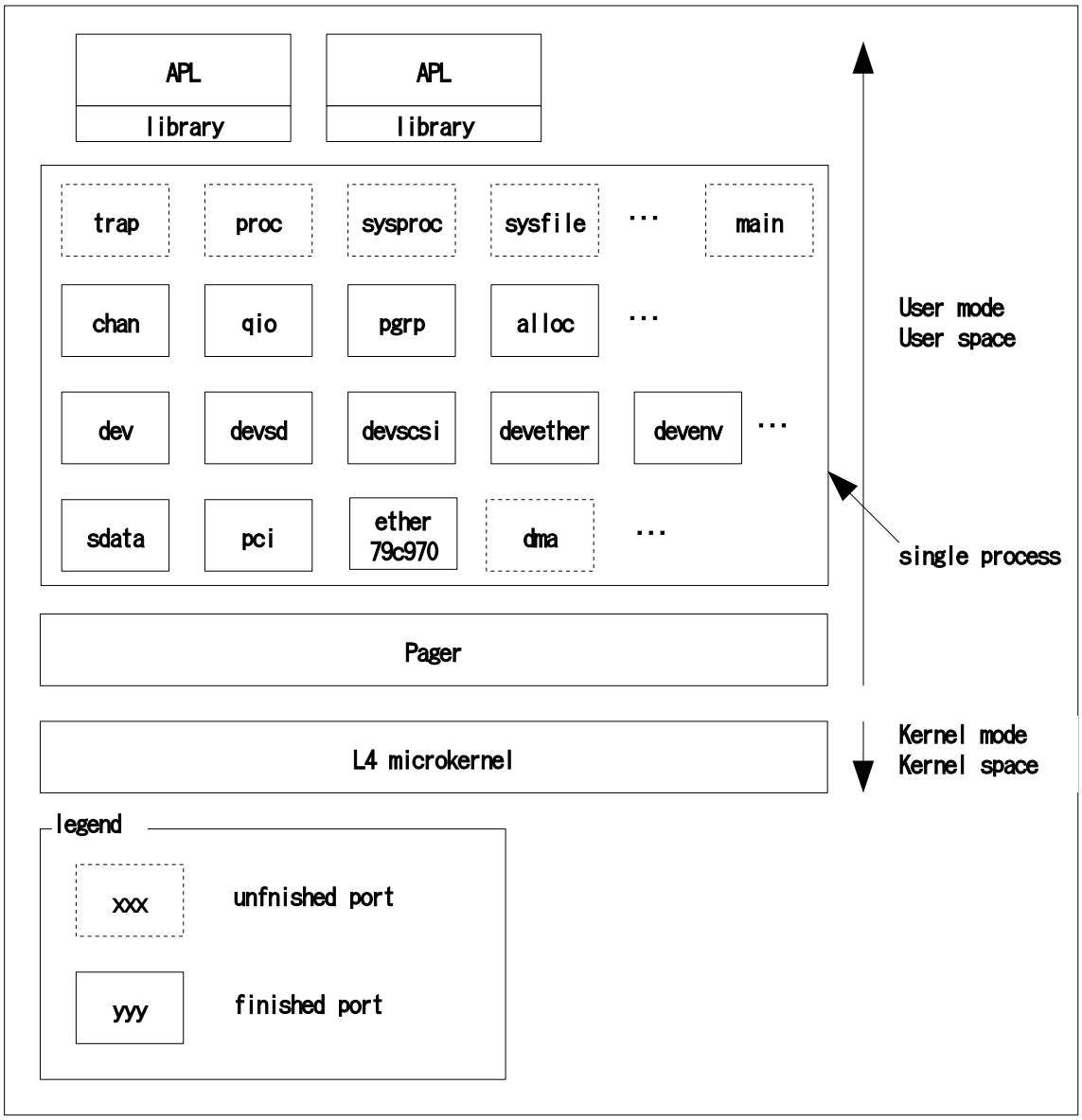

Figure 6.1 Single server version of LP49

Figure 6.1 shows the current status of LP49. We haven't separated the PM and RMs as of yet. The most fundamental functions are starting to work, including an HDD driver and an Ether driver (for AMD 79C970).
We are implementing the system calls.

7 Related research

- Plan 9 – component-oriented distributed processing OS

- L4 – a microkernel with an efficient and flexible IPC and thread

  As stated above, LP49 is a Plan 9-based OS on the L4 microkernel.

- Plan B – a Plan 9- based OS with a box abstraction instead of a file with ubiquitous environment

  Plan B put resources int boxes, and users can transparently access both local and remote resource. Of cource, every process has its own name space, as same as Plan 9.

- QNX – a commercial microkernel OS for embedded systems

  The concept of LP49 is similar to that of QNX. Developing with QNX may be difficult because it's a commercial OS and its source code is not open.

- Symbian OS– a popular specialized OS for cell-phones

  Most of the Symbian OS is implemented in C++. Thus, applications can be developed using object-orientation, and they are easily ported.

8 Future work and Conclusion

The Process Manager and Resource Managers are note separated in the current version that we are now implementing. In the next step of our development, we will try to separate them into a PM and several RMs, and do a complete evaluation.
Hence, new functions added to LP49 won't affect the existing functions. For example, when you add a new device driver to LP49, you can create a new process for the device driver. Of course, the other OS services are independent from the new process.

We think the collaboration between Plan 9 and the L4 microkernel will be useful in embedded systems.