情報処理学会論文誌 Vol. 50 No. 2 1234–1243 (Feb. 2009)

分散OS学習開発キット LP49

丸山勝巳[†1] 佐藤好秀[†2]

An open source kit for studying and developing a distributed OS "LP49"

Katsumi Maruyama[†1] and Yoshihide Sato[†2]

Control systems (e.g. embedded systems, home servers) are becoming more and more sophisticated, networked and complex. Software productivity and dependability are the first concern of their design. OS 's for control systems must support distributed processing, fail-robustness and ease of program development. LP49 is a component-oriented OS with micro-kernel and multi-server architecture. We have adopted the L4 micro-kernel because of its performance and flexibility. Plan 9 had devised excellent distributed processing facilities (e.g. 9P protocol, private name space, user-mode servers), and we have largely adopted concepts and source code from Plan 9. LP49 component architecture will be effective to improve dependability.

1. はじめに

今や組み込みシステムやホームサーバには、NW 接続された多様な機器を連携動作させる分散処理機能が必須である。OS はシステムの性能 (機能・効率・信頼性) のみならずプログラム開発コスト (開発しやすさ、開発期間・拡張性など) を大きく左右する。信頼性向上の近道はシンプル化であり、目的に最適な分散 OS を作りたい人にはシンプルな分散 OS 開発キットは意義がある。

†1 国立情報学研究所
NII
†2 日立製作所
Hitachi

また、OS はソフトウェア技術の集大成であり、最高のソフトウェア教材である。学習用 OS は、重要機能を含み、構成が簡明で、容易に機能追加でき、プログラム規模も小さいことが望まれる。学習用 OS としては、Minix が大変すぐれているが、残念ながら分散処理機能は目的としていない。

このように、分散 OS の適切な開発学習キットは大変意義が高い。本報告は、以上の観点から分散 OS 開発学習キット LP49 について紹介する。

2. 本 OS 開発学習キットの意図

シンプルな分散 OS 開発学習キットに向けて以下を目指している。

制御システム (組込みシステム) やサーバに適したシンプルなＯＳ 制御用、サーバ用に必要な機能をもったコンパクトな分散 OS。

障害に対する頑強性の強化 → マイクロカーネル＋マルチサーバ構成 モノリシック OS では部分障害 (例えばドライバのバグ) でもシステムクラッシュを導きがちである。障害が生じても波及範囲を閉じ込めて部分再開を可能とするためには、マイクロカーネル＋マルチサーバ構成の OS が有利である。

拡張性の強化 機能追加はユーザモードプロセスの追加などにて容易に行えるようにする。

システム連携の機能 今後の組込みシステムには、分散リソースの体系的な管理と制御、ノード間での名前空間の可視化、環境変数を含む動作環境の連携など従来の分散 OS 以上の機能が要求されよう。

また、分散処理のプロトコルは、提供できる機能と性能を決定する。独自プロトコルは強力であっても普及させることは難しい。Plan9 の **9P プロトコル**は、`attach()`, `walk()`,`open()`, `create()`, `read()`, `write()`, `clunk()`, `remove()`, `stat()`, `wstat()` 等のメッセージからなり、低レベル制御も可能で融通性が高いので、これを採用する。

プログラム開発の容易化 モノリシック OS カーネルは、カーネル (特権) モードで動いている。カーネルモードプログラムは、開発に高いスキルが必要でデバッグが大変難しい。その上, 小さなバグでもシステム全体がクラッシュしかねない。本 OS では、マイクロカーネル以外は全てユーザモードプログラムとし、ほとんどのサービスはユーザモードプロセスの追加で実現でき、デバインスドライバもユーザモード化する。これにより、プログラム開発が容易化・効率化される。

L4 と Plan9 のソースコード活用 OS 全体をスクラッチから作るには、膨大な工数を要する。Karlsruhe 大学の L4 マイクロカーネルは、簡潔で優れたスレッド・メッセージ性

能を持っている。また、Bell 研で開発された Plan9 は、融通性の高い分散処理を実現している。かつ両者ともオープンソースであるので、ソースコードを活用して工数削減をはかった。

ソースコードの公開と使い慣れたプログラム開発環境　OS を学習したり自前の OS を開発したい人に手頃なソースコードを提供する。また, 使い慣れた GNU 環境でコンパイルからテスト走行まで出きるようにする。

3. プログラム構成

LP49 は、図 1 に示すように以下の階層からできている。

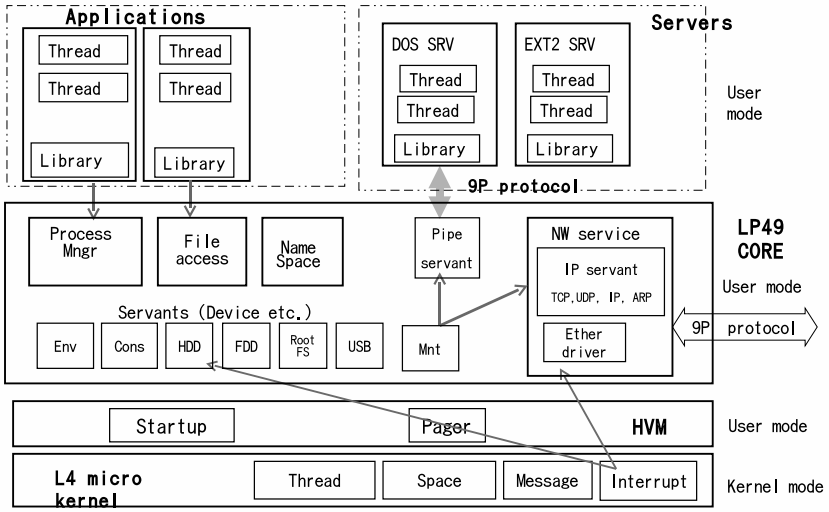


図 1　LP49 全体構成

マイクロカーネル階層 (カーネルモード)　L4 マイクロカーネルそのものであり、ここだけがカーネルモードで動作している。マルチスレッド、タスク (=プロセス) 論理空間、スレッド間通信、ページマップの諸機能を提供している。

HVM 階層 (ユーザモードプロセス)　ユーザモードで走る L4 プロセスである。LP49 の立ち上げ、スレッド制御と論理空間制御、並びにページフォールトの対処を行うスレッド (`Pager`) を持っている。L4 マイクロカーネルは、スレッドの実行中に pagefault が生じると、本 **Pager** スレッド に Pagefault メッセージを送る。`Pager` は pagefault メッセージを受信すると、L4 の page map 機能を使って適切なページを割り当てる。なお、応用プログラムは, page fault の処理を HVM の `pager` に行わせても良いし、自分独自の Pager を記述することも可能で、ページキャッシュ、トランザクションメモリなど融通性の高いメモリ管理も実装できる。

HVM という名前は、将来 Hypervisor Monitor 階層として Virtual Machine に発展させることを考えて付けたが、現時点ではその機能は持っていない。

Core 階層 (ユーザモードプロセス)　APL プロセスからシステムコール (実際にはメッセージ) を受けて、所望の処理を行う。また、必要に応じてサーバプロセスに処理を依頼する。つまり、プロセスのシステムコールに対してはサーバ、サーバプロセスに対してはクライアントとして機能する。プログラムは、ユーザモードつまり非特権モードで走るので、安全性が高い。デバイスドライバもここに含まれる。

サーバ階層 (ユーザモードプロセス)　OS サービスを行うファイルサーバー類も普通の応用プログラムも、同等のユーザモードプロセスである。ファイルサーバは、後述の様に 9P プロトコルを喋れる点がちがうだけである。

4. サービスと抽象ファイル

4.1 サービス提供：サーバとサーバント

OS が提供するサービスには、Dos ファイルサービス、Ext2 ファイルサービスといった高位サービスと、ハードウェア駆動、モジュール間通信、NW 接続、サーバ登録簿といった低位サービスとがある。

前者は、個々に独立性が高く、規模も大きくなりがちなので、サービス毎にユーザモードで走るプロセスとして実現する。これを**サーバ**と呼ぶ。サーバはメッセージインタフェースなので、ローカルでもリモートでも同等に使える。ユーザモードプロセスなので、プログラム開発の容易化のみならず、障害時もそのサーバだけを停止・再開することで耐障害性も強化される。

後者は共通機能的で、より実行速度が重視されるので、独立したプロセスとはせず LP49core 内のモジュールとして実装することとした。このモジュールを**サーバント**と呼んでいる。サーバントは、統一インタフェースを持つコンポーネントである。機能的には、同一サービスをサーバとして実装することもサーバントとして実装することも可能である。

サーバもサーバントも、内部要素を登録する名前空間 (directry tree) を持っており、プロセスはそれを自分の名前空間にマウントすることにより、普通のファイルインタフェースで目的要素にアクセスして、内容の読み書き・制御を行える。

(1) サーバント

サーバントはハードウェアデバイス, サーバ登録簿、環境変数記憶、pipe, プロトコルスタックなど低位サービスを提供する。サーバントは LP49core プロセス内のモジュールであり、attach(), init(), open(), read(), write(),,, といったプロシージャインタフェースで呼ばれる。

各要素はサーバントの名前空間に登録されており、ファイルインタフェースで操作できる。サーバントは #+<英字> の形式のサーバント識別子をもつ。代表的サーバントを以下に示す。括弧内は サーバント識別子である。

コンソール (#c), ハードディスク (#S), フロッピーデバイス (#f), 環境変数 (#e), サーバ登録簿 (#s), サーバマウント (#M), Ether ドライバ (#l), プロトコルスタック (#I), Pipe(#|), ルートファイルシステム (#R), USB ホストコントローラ (#U), VGA コントローラ (#v),,,

"bind" コマンドにより、サーバントをプロセスの名前空間に結合することにより、プロセスからサーバントにアクセスできるようになる。

```
LP49[/]: bind [-abc]  サーバント名　マウントポイント
```

(2) サーバ

サーバはサービスごとに独立したユーザモードのプロセスであり、9P メッセージ (9P プロトコル) を受信してサービスを実行する。LP49core とサーバの間で 9P メッセージを運ぶ接続を**サーバリンク**と呼んでいる。サーバリンクは、ローカルサーバの場合は pipe (LP49 の pipe は 双方向である)、リモートサーバの場合は TCP/IP 接続を用いる。

サーバは自分のサーバリンクを "サーバ登録簿" に登録しておく。サーバ登録簿はサーバントの一つ (#s) で、立ち上げ時に "/srv"として接続されている。クライアントは、サーバ登録簿から目的のサーバを見つけて、サーバリンク名 (ex. /srv/dos) を自分の名前空間にマウントする。これにより、サーバの名前空間がクライアントの名前空間に接続され、普通のファイルインタフェースでアクセスできるようになる。

```
LP49[/]: mount [-abc]  サーバリンク名　マウントポイント　付加指定
```

4.2　抽象ファイルオブジェクト

Plan9 と同様、LP49 では殆どのリソースを "ファイル" として抽象化している。個々の "抽象ファイル" は、サーバント内あるいはサーバ内に存在し、名前空間 (directory tree) に登録されている。

オブジェクト指向の観点からは、図 2 に示すように "抽象ファイル" はインスタンスオブジェクト、"サーバント定義" はクラス定義に相当する。各サーバントは同一のインタフェースを有する。

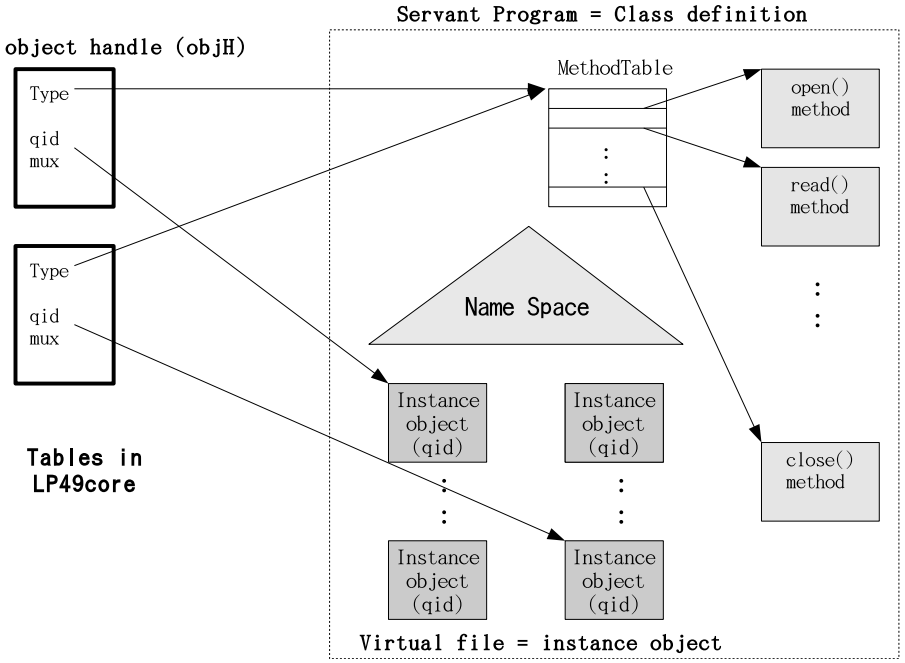


図 2　仮想ファイルのオブジェクトモデル

- 各サーバントプログラムはクラス定義に相当し、attach(), init(), open(), read(), write(),,, 等のメソッドコードを持ち、メソッドテーブルを介して呼ばれる。
- “抽象ファイル” は、インスタンスオブジェクトであり、サーバントの名前空間に登録されている。以下では **VF(抽象ファイル) オブジェクト**と呼ぶ。
- VF オブジェクトは、qid という値によりサーバント内でユニークに識別される。Qid は Unix の inode 番号に相当する。
- 一般のオブジェクト指向言語と違って、同一クラス (サーバント) 内でも VF オブジェクトのデータ構成は同一とは限らない。各メソッドは VF オブジェクトの qid から、そのデータ構成を判定し、対応した処理を行う。

LP49core 内では、VF オブジェクトは**オブジェクトハンドル**を介してアクセスされる。オブジェクトハンドル (objH) は、サーバント識別情報 (type フィールド), VF オブジェクトの qid (qid フィールド), その他の管理情報が載ったテーブルである[*1]。LP49core は、オブジェクトハンドルの type フィールドからサーバントの各メソッドをアクセスし、qid フィールドからインスタンスを決定する。ここでは VF オブジェクト α を指すオブジェクトハンドルを “objH{α}” と表記する。

オブジェクトハンドルは、対象を open(), create() した時や、change directory された時に割り当てられ、参照カウンタが 0 になったときに消去される。

5. システムコールの仕組み

5.1 サーバントによる処理

システムコールは、モノリシック OS では一般にトラップを用いるが、マイクロカーネル OS ではメッセージ通信を用いる。システムコールの仕組みを図 3 に示す。

(1) ライブラリによる L4 メッセージ化

APL のシステムコールは, 使い慣れた関数呼び出し (open(...), read(...), write(...) ,,,) である。ライブラリは、これを L4 メッセージに変換して LP49core に送り、返答メッセージを待つ。APL と LP49core は別論理空間なので、アドレス引き継ぎは使えない。システムコールの引数引き継ぎには、小サイズのデータは L4 メッセージの値コピー, バッファー域は L4 メッセージのページマップ機能を利用した。

(2) LP49core マルチスレッドサーバ

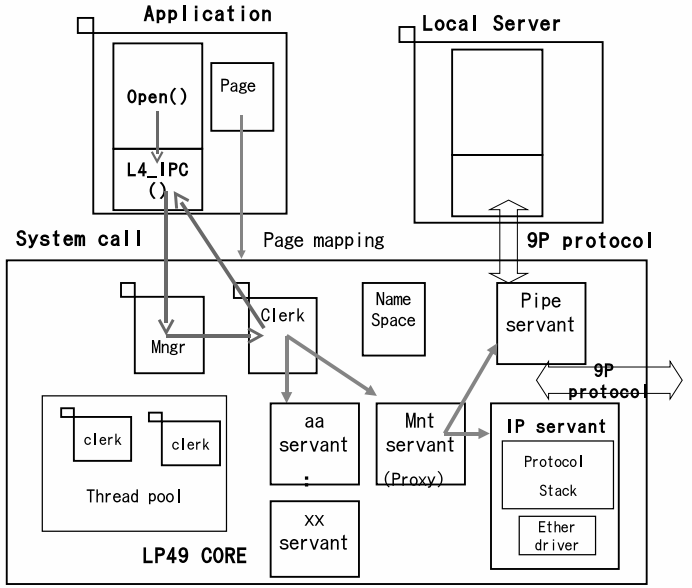


図 3　システムコール

LP49core は、APL のシステムコール処理に対してはサーバ、サーバプロセスの 9P メッセージ処理に対してはクライアントとして機能する。システムコール処理は中断が生じうるので、複数要求を並行して処理するためにマルチスレッドサーバを実装した。要求メッセージは Mngr スレッドに送られる (L4 メッセージの宛先はスレッドである)。Mngr スレッドは、スレッドプールから空き clerk スレッドを割り当てて、処理を行わせる。

(3) 目的処理の実行

プロセス制御の場合には、プロセスマネジャーに処理を行わせる。

ファイル処理の場合には、サーバントが処理を行う。サーバントを接続すると、そのルートを表す objH テーブルが割り付けられる。chdir すると新場所を表す objH テーブルが割り付けられる。ファイルを create あるいは open すると、その objH テーブルが割り付けられる。オブジェクトの操作をする場合、その objH テーブルにアクセスして、サーバント

⋆1 Plan9 のソースコードを流用したので、プログラム上では Chan(nel) タイプのテーブルとなっている

(クラス) とオブジェクトを決定し、サーバントのメソッドを呼び出して処理を行わせる。

分散処理の説明は、後に説明する。

6. 分散処理と名前空間

6.1 サーバのマウント

(1) サーバリンク

サーバと LP49core の間で 9P メッセージを運ぶ接続がサーバリンクである。サーバリンクは、ローカルサーバの場合は pipe, リモートサーバの場合は TCP 接続である。

Pipe は pipe サーバント (`#|`) のオブジェクト、TCP 接続は IP サーバント (`#I`) のオブジェクトであり、LP49core 内では `objH` テーブルによって表現される。

(2) サーバ登録簿

サーバ登録簿 (`#s`) は目的サーバのサーバリンクを見つけるためのものであり、初期設定により名前空間の “`/srv`” に接続されている。各サーバは、サーバリンクをサーバ登録簿サーバントに登録する。例えば “`/srv/ext2`” には、Ext2 ファイルサーバのサーバリンクが記録されている。

クライアントは、サーバ登録簿から目的のサーバを見つけて、これを自分の名前空間にマウントすることで、サーバの持つ名前空間にアクセスできるようになる。

(3) サーバマウント

例えば、次のコマンドは DOS ファイルサーバ (`/srv/dos`) を使って、HDD (`/dev/sdC0`) の DOS partition を `/c` にマウントする。

[例] `LP49[/]: mount -a /srv/dos /c /dev/sdC0/dos`

サーバマウントの要となるのが、マウントサーバント (`#M`) である。マウントにより、マウントポイントを指す “`objH{` マウントポイント `}`” テーブルが割り当てられ、`type` フィールドにはマウントサーバントが、`mchan` フィールドには objH{ サーバリンク } が、`qid` フィールドには qid 値が設定される (つまりマウントサーバントのオブジェクト)。

`objH{` マウントポイント `}` に対する操作は、マウントサーバントによって 9P メッセージに変換され、サーバリンクを通してサーバに送られ、そこで所望の処理が行われる。つまり、本 `objH` テーブルはサーバ内の本体オブジェクトの **Proxy オブジェクト**として機能している。

同様にサーバ上のオブジェクトを `create` あるいは `open` すると、それに対応した Proxy オブジェクトが割り当てられる。このようにしてサーバ上のオブジェクトもローカルと同じように操作できる。

6.2 サーバアクセス処理の具体例

サーバの “`/`” をプロセスの名前空間 “`/c`” にマウントし、“`/c`” 及び “`/c/x/y`” にアクセスした場合の処理内容を図 4 に示す。

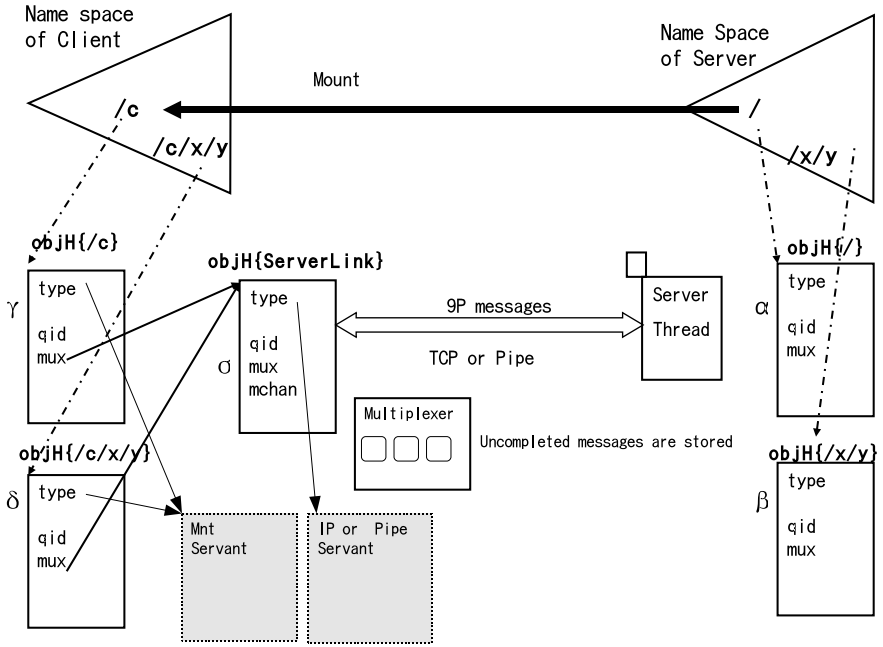


図 4　リモートオブジェクトアクセス

(1) サーバ登録簿への登録: サーバ登録簿にサーバリンクを登録することにより、“`objH{` サーバリンク `}`” (図の σ) が割り当てられる。

(2) サーバのマウント: `mount` コマンドによりサーバをマウントする。

`LP49[/]: mount -ac /srv/dos /c /dev/sdC0/dos`

`/srv/dos` はサーバリンクを, `/c` はマウントポイントを, `/dev/sdC0/dos` はサーバ名前空間のマウント開始位置を、`-ac` は書き込み可能で after にマウントすることを意味する。

これにより、サーバとの間で attach メッセージなどのやり取りを行い、“objH{/c}” (図の γ) テーブルを割り当て、以下の設定を行う。

- type フィールドには Mnt サーバントを設定。つまり、objH{/c} のクラスを Mnt サーバントである。
- mchan フィールドには objH{ サーバリンク } を設定。
- qid フィールドには サーバからもらった qid を設定。この qid は、サーバ内のマウント開始位置オブジェクトの qid である。

以上により、objH{/c} は、サーバ内のマウント開始位置オブジェクト (図の α) の **Proxy オブジェクト**となる。こうして、objH{/c} への操作はマウントサーバントによって **9P メッセージ**に変換され、サーバリンクに送り出される。サーバからの返答を受けたらそれをクライアントに返す。

(3) サーバ名前空間での操作: 例えば /c にて “open("x/y", OREAD)” を実行したとする。この場合は、サーバとの間で walk, open メッセージなどをやり取りして、“objH{/c/x/y}” (図の δ) テーブルを割り当て、以下の設定を行う。

- type フィールドには マウントサーバントを設定。
- mchan フィールドには objH{ サーバリンク } を設定
- qid フィールドには サーバからもらった qid を設定。この qid は、サーバ内の /b/c/d オブジェクト (図の β) の qid である。

以上の設定により、objH{/c/x/y} は、サーバ内の/x/y オブジェクトの Proxy オブジェクトとなる。

6.3　離れた名前空間の可視化

リモートノードの任意の部分空間 (そこには複数のサーバントやサーバが接続されていてもよい) を、自プロセスの名前空間にマウントすることができる。Plan9 から移植した extfs サーバと import コマンドが、これを実現する。

extfs サーバは、指定された部分名前空間を外部ノードに export するサーバ、import コマンドはそれを自分の空間にマウントするコマンドである。

例えばリモートホストの /dev ディレクトリーをローカルホストにマウントすると、/dev に接続されているリソースにアクセスすることが可能になる。全てのオブジェクトはファイルインタフェースで操作できるので、このことはリモートホストのデバイスも操作できることを意味する。

同様にして、U9FS というプログラムを Unix 上で走らせることにより、UNIX のファイルシステムの部分空間を LP49 上にマウントすることも可能である。U9FS は UNIX で 9P プロトコルを理解するプログラムで、Plan9 ユーザグループから移植した。

6.4　名前空間の機構

図 5 に示すように、サーバントやサーバは、ルートファイルシステム **RootFS**(これもサーバント) を出発点とする名前空間に接続（マウント）することにより、プロセスに見えるようになる。Plan9 同様に、同一マウントポイントに複数をマウントすることが可能である (union マウント)。“/dev” には各種デバイスサーバントが、“/net” にはプロトコルスタックが接続されている。

UNIX ではファイルシステムの mount は root 権者のみが行え、名前空間は全プロセスで共通である。これに対し、Plan9/LP49 の名前空間は、各プロセス毎に自前の名前空間（**個別名前空間**）を持つことができる。名前空間に見えないものはアクセスできないので、緻密なセキュリティー管理を実現できる。

名前空間の構成法を図 6 に示す。図中の (a) では、指定されたマウントポイントにサーバントを結合 (bind) することにより、サーバントの名前空間がプロセスの名前空間に接続され、サーバントのサービスを受けられるようになる。

同様に (b) では、サーバーをマウント (mount) することにより、サーバの名前空間がプロセスの名前空間に接続され、サーバのサービスを受けられるようになる。サーバは remote procedure call で呼び出されるので、自ホスト内 (b) でもリモートホスト上 (c) でも、同様にアクセスできる。

また、(d) は extfs サーバと import コマンドをにより、別ノードの “部分” 名前空間をマウントしている。

6.5　プロトコルスタック

Plan9 のプロトコルスタックはモジュール化されており、比較的軽い修正で LP49 に移植しえた。

インターネットサービスは、図 7 に示すように IP サーバント (“#I”) が提供している。IP サーバントの下では、TCP, UDP, IP など各プロトコル毎にモジュール化されたプログラムが動作しており、ユーザには以下のトリー構成をもつファイルシステムとして見える。IP サーバントは “/net” に接続され、個々の接続もファイルインタフェースを持つ。例えば TCP 接続は、/net/tcp/0, /net/tcp/1 ,,, として名前空間に見える。

```
--+- tcp/ ----+- clone
  |           |- stats
```

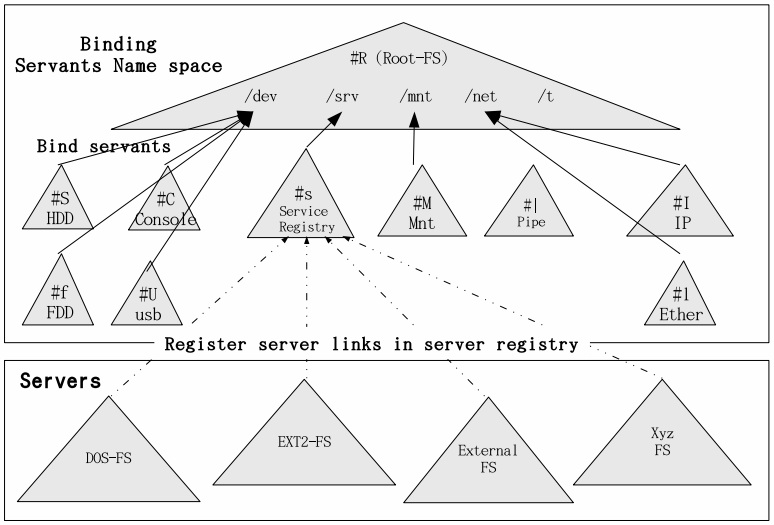


図 5　サーバ、サーバントと名前空間

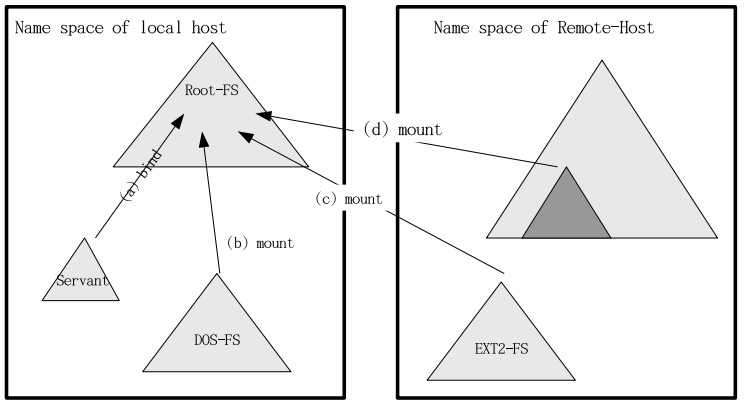


図 6　名前空間とマウント

```
|           |- 0/ ---+- ctl
|           |        |- data
|           :        |- local
|           :        |
|
|- udp/ ---+- clone
|          |- stats
|          |- 0/ ---+- ctl
|          |        |-data
|          :        |-local
|          :        |
|
|-- arp/ ----
```

Ether サーバント (`#l`) は、Ether card のサービスインタフェースであり、該当した Ether driver を呼び出している。

7. 本 OS キットの開発環境展

OS の開発および学習には、使い慣れた開発環境が望まれる (Plan9 が普及が遅れている理由には独自の開発環境もある)。また OS の走行テストも、NW 機能を含めて実機の前に仮想マシンで行えることが望まれる、

LP49 のプログラム開発は Linux ホスト上の GNU ツールだけで、LP49 の走行テストはオープンソースエミュレータ Qemu で行っている。Qemu はネットワーク機能も提供しており、1 台のホストマシン上で、NW 接続した複数の LP49 を走らせることも、LP49 と Linux ホスト間を NW 接続するとも可能である。1 台のホストマシン上で、コンパイルから NW を含む走行試験まで時間を待たずに (全 make でも 30 秒、LP49 立ち上げに 10 秒)

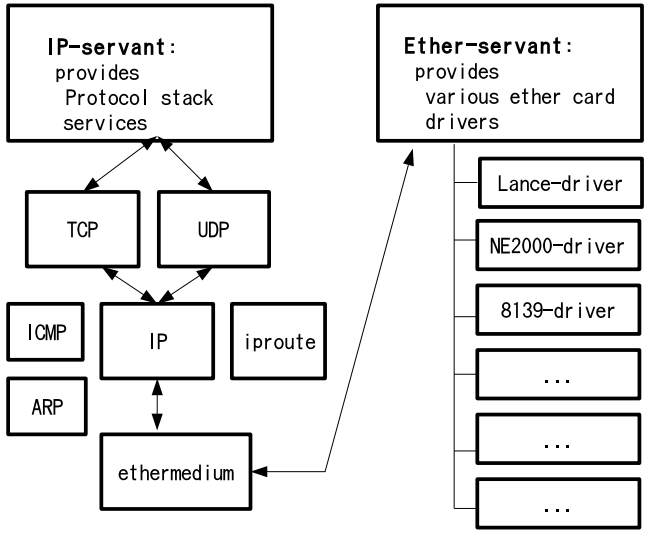


図 7　IP サーバントと Ether サーバント

できるので、大変快適である。

8. 本 OS キットによる展開例

(1) サーババス

LP49 では、OS サービスはサーバもしくはサーバントによって提供される。LP49core は、サーバント・サーバのマルチプレクサと APL とサーバやサーバとの間を連携させるメッセンジャーに徹しており、これがシンプル化に結びついている。ただし、現方式ではシステムコールは全て LP49core を経由して処理されるが、目的サーバが決まってしまえば APL とサーバの間で直接メッセージをやり取りさせることもできる。具体的には、目的サーバの決定に関わる open(), create(), chdir()、close() を LP49core が処理して APL-サーバ間を安全なチャネルで接続させる。つまり OS はサーバを接続するための広域バスとすることが可能で、この形態が本研究の最終目的である。

また、9P プロトコルは同期型メッセージであるが、広域分散をより効率的に行うためには非同期型メッセージのサポートが有効と思われる。

(2) 耐障害強化

プロセスを監視し、障害が見つかればそれを停止して再スタートする仕組みを容易に組み込める。また、簡単な Pager の機能追加により、プロセスに保持メモリ域を追加することができる。保持メモリ域には、プロセスが指定した適当なタイミングで無矛盾なデータのスナップショットを書き込んでおく。プロセスが障害になった場合には、保持メモリ域のデータを使って再開させることにより、比較的安全な roleback を行える。

(3) 高性能化

現 LP49 には、高性能化の仕組みは殆ど組み込んでない。スレッドの優先度制御、ページキャッシュ、システムコール引数引き継ぎ等の改良で容易に高性能化を図ることができる。Pager は作り直しを予定しており、この際に簡単なページキャッシュを実現する予定である。

8.1 認証システム

分散 OS において認証は非常に重要な課題である。Plan9 は Kerberos を拡張した精妙な認証方式を実装しているが、学習用としては複雑である。LP49 では,Identity-Based Encryption を使用した認証方式を検討している。

9. 関 連 研 究

分散処理機能の観点からは、LP49 は Plan9 を延長した学習開発実験キットとも云える。Plan9 のソースコードをできる限り活用することとしたが、実際にはかなりの変更を必要とした (表 1)。マイクロカーネル化に伴い、プロセス・スレッド機能とシステムコール機構は全面的に変更を要した。プログラム構造的には、マイクロカーネルによる障害保護強化、サーバントによる部品化強化などを行った。また、Plan9 のサーバとドライバプログラムを少しの修正で移植できる。9P プロトコルの採用により、少ない修正で Plan9 のサーバを LP49 に移植できるという利点もある。

Plan9 が魅力的な内容にも関わらずハッカーが限られているのは、オープンソース化の遅れとともに、独自の C 言語と開発環境が理由と思われる。特に構造体の無名フィールド (フィールド名を省略でき、その場合コンパイラがデータタイプから目的フィールドを自動的に探す）が多用されており、ANSI コンパイラに通らないのみならず、プログラム解読も難しくしている。LP49 では、プログラム修正も行い GNU 環境で開発できるようにした。[⋆1]

L4 は, LP49 が採用したマイクロカーネルである。使い方も容易で、何ら機能追加や修正

⋆1 GCC への Plan9-C 仕様の追加も考えたが、GCC は改版が頻繁なので断念した.

表 1　LP49 と Plan9 の対比

分類	Plan 9	LP49
Micro kernel	No	Yes
並行処理	プロセスはコルーチン、 スレッドはメモリ域共用のプロセス	L4 Process L4 Thread
システムコール	Trap カーネル+ユーザプロセス	L4 メッセージ マルチスレッドサーバ
〃データ入力 〃データ出力	Plan9 カーネルが APL 空間を直接アクセス	L4 ページマップ L4 ページマップ
ドライバ	カーネルモード	ユーザモード
言語仕様	Plan9 独自の C 無名フィールド, 無名パラメータ typedef, USED(), SET() #pragma, 自動ライブラリリンク	GCC
コンパイラ	Plan9 独自 C コンパイラ	GCC
Utility	Plan9 の Linker, Assembler, mk	GCC, gld, gmake
Binary	a.out 形式	ELF 形式

をすることなく活用できた。

Minix は学習用マイクロカーネル OS として、非常に意義が高い。Linux は Minix から影響を受けて開発されたが、マイクロカーネルは採用されなかった。全カーネルデータが見えるモノリシック OS の方が作り易いし効率も良いという主張に一理はあるが、障害に対する頑強性、プログラム保守性はマイクロカーネルが有利である。Minix は分散 OS 機能は範囲外である。

SawMil は IBM で実施された L4 マイクロカーネルとマルチサーバからなる OS の研究プロジェクトである。Linux カーネルをサービス対応にモジュール分けしてマルチサーバ化することを狙ったが、Linux カーネルはモジュール分割が困難のため、プロジェクトは中座した。

L^4Linux は、L4 マイクロカーネルの上で Linux を動かす OS である。Linux はモノリシック構成のままであり、L4 を使った Virtual Machine といえる。

GNU の **Hurd** は、古くから挑戦しているマイクロカーネル OS である。マイクロカーネルとしては **Mach** を採用していたが、効率の観点から最近は L4 マイクロカーネルを採用した **L4-Hurd** を検討している。Unix 互換が目標であり、分散 OS を目指したものではない。

10. おわりに

LP49 の詳細な資料とソースコードは、WEB サイト http://research.nii.ac.jp/H2O/LP49 にて公開している。

(1) 学習用 OS としての観点

コメントを含むソース規模は、HVM: 約 2 K 行、LP49core: 約 60K 行 (内 20K 行がプロトコルスタック)、libc ライブラリ: 約 30K 行、その他のライブラリ:x K 行、rc シェル: 約 8K 行、デバッグシェル: 2K 行、DosFS: 40K 行、Ext2FS: 30K 行、extfs: 2K 行である。機能に比べて充分にコンパクトであり、一人で全てをトレースできる。本キットにより、LP49 だけでなく Plan9 と L4 のプログラム技術も理解できる。

(2) 分散 OS 開発キットとしての観点

現 LP49 の測定値は以下の通りである (実機は Pentium3 1GHz, Qemu 走行は Pentium4 2.8GHz, Dedora8)。

Qemu 上の LP49 – LP49 間の ping 時間： 4.7ms. Qemu 上の LP49 – Linux/host 間の ping 時間： 0.7ms. Qemu 上の LP49 – LP49 間のリモートファイル (1KB) 読み出し時間： 40ms. Qemu 上の LP49 の CD ファイル (1KB) 読み出し時間： 690 μ sec.(初回は 25ms). 実機上の LP49 の CD ファイル (1KB) 読み出し時間： 120 μ sec.(初回は 110ms). Qemu 上の LP49 の RAM ファイル (50B) 読み出し時間： 180 μ sec. Qemu 上の LP49 の RAM ファイル (4KB) 読み出し時間： 220 μ sec. 実機上の LP49 の RAM ファイル (50B) 読み出し時間： 31 μ sec. 実機上の LP49 の RAM ファイル (4KB) 読み出し時間： 48 μ sec.

まだ最適化は行っていないので性能的には充分とはいえないが、前述のスレッド優先度制御、ページキャッシュなどを導入することで、性能は向上できる。

大部分の機能追加はサーバ追加で行えること、サーバはユーザプロセスなので開発が容易なことなどの利点がある。

(3) 今後の予定

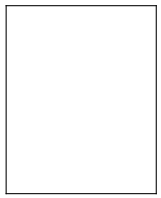
公開ソースコードのリファインを進めて、より簡潔で理解し易くするとともに、障害プロセスの停止再会機構と最適化を進める予定である。そのあと更に簡潔な広域サーババスの実装を行う。

謝辞　..... の皆様に，謹んで感謝の意を表する．

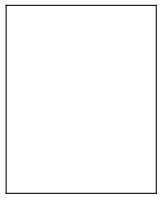
参 考 文 献

1) J. Lidtke: *On micro-Kernel Construction*, Proc. of IEEE International Workshop on Object-Orientation in Operating Systems (IWOOOS), (1996).
2) Karlsruhe univ. : *L4 Web site*, http://l4ka.org/ .
3) Rob Pike, et. al. ： *Plan 9 from Bell Labs*, ... (1991).
4) Bell labs.： *Plan 9 Web site*, http://plan9.bell-labs.com/plan9/
5) A. Tannenbaum & A. Woodhull： *Operating systems design and implementation,3/E*, Addison-Weisley, (2000).
6) www.minix3.org.：*Minix Web site*, http://www.minix3.org/
7) H_2O： *LP49 Web site*, http://research.nii.ac.jp/H2O/LP49/
8) N. C. Hutchinson & L. L. Peterson: *The x-Kernel: An architecture for implementing network protocols*, IEEE Transactions on Software Engineering, 17(1), Jan. 1991.
9) Joe Armstrong: *Making reliable distributed systems in the presence of hardware errors*, 博士論文 (Ph.D.) 、スウェーデン王立ストックホルム工科大学, 2003.
10) G. Hunt. et. al.: *An overview of the Singularity Project*, Microsoft research technical report MSR-TR-2005-135, 2005.

(平成 20 年 9 月 17 日受付)
(平成 20 年 11 月 28 日採録)



丸山　勝巳（正会員）
.....



佐藤　好秀（正会員）
.....