論　文

分散 OS 開発学習キット LP49

丸山　勝巳[†]　　佐藤　好秀[†*]

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

Katsumi MARUYAMA[†] and Yoshihide SATO[†*]

あらまし　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．信頼性向上の鍵はシンプル化であり，目的に最適なシンプルな分散 OS の開発・学習キットは意義がある．開発・学習用 OS は，重要機能を全て含み，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．本報告では，マイクロカーネルとマルチサーバを基本とした分散 OS 学習・開発キットについて紹介する．

キーワード　分散 OS, L4, Plan9, マイクロカーネル，マルチサーバ，コンポーネント

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．信頼性向上の鍵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．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．学習用分散 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.

マイクロカーネル＋マルチサーバ構成による障害に対する頑強性の強化: 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: 最近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 を越えた機能が要求される．

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が，独自プロトコルは普及させることは難しい．Plan9 [1] の **9P プロトコル**は，`attach()`, `walk()`, `open()`, `create()`, `read()`, `write()`, `clunk()`, `remove()`, `stat()` 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，これを採用した．

プログラム開発の容易化: モノリシック OS 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

[†] 国立情報学研究所
National Institute of Informatics
[*] 日立製作所

イスドライバもユーザモード化した．これにより，プログラム開発が容易化・効率化される．

L4 [3] と Plan9 [1] 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．

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とともに, 使い慣れた GNU 環境でプログラム開発からテスト走行までできるようにした．

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．

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，唯一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 page fault 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，適切な実ページを決定し L4 の page map 機能を使って割り当てる．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意図したが，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，必要ならばサーバプロセスに仕事を依頼して，要求された処理を行う．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 プロトコルを話せ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プロセスとして実現した．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，内部リソース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リソースにアクセスして，内容の読み書き・制御を行える．プログラム構造的にも優れたコンポーネントと言える．

(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

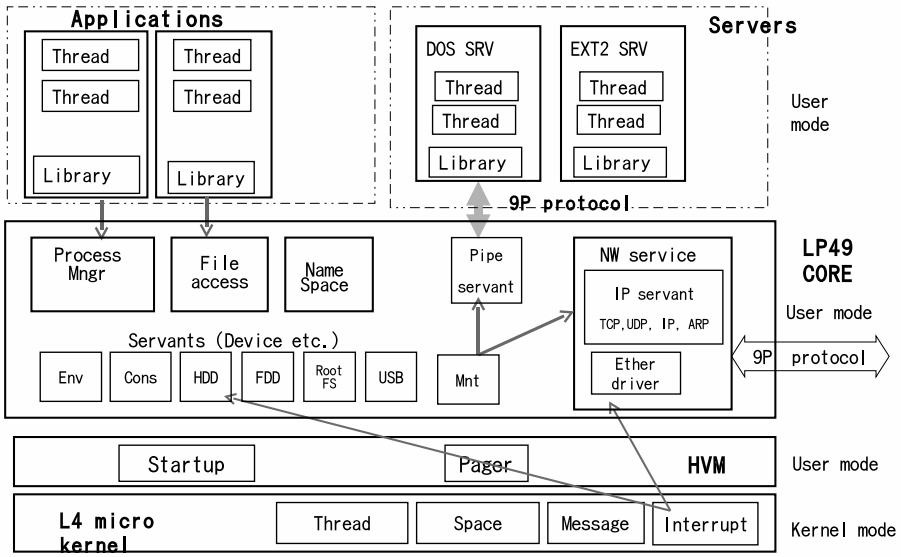


図 1　LP49 全体構成
Fig. 1　LP49 structure

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．

bind サーバント識別子 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．各サーバのサーバリンクを登録しておくデータベースが **サーバ登録簿**である．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．

mount サーバリンク マウント位置 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．

- 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．

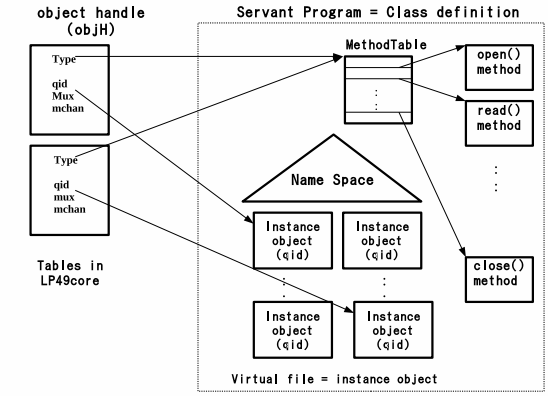


図 2　仮想ファイルのオブジェクトモデル
Fig. 2　Virtual file object model

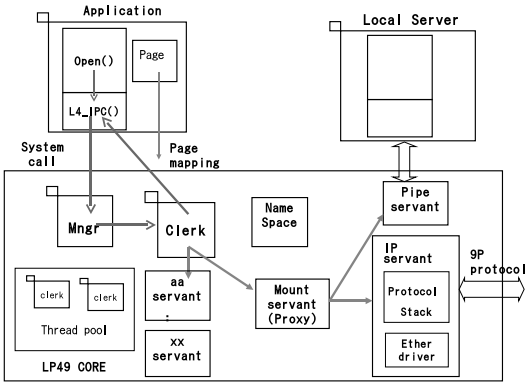


図 3　システムコール
Fig. 3　System call

- “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) 内でも各 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** (`objH`) を介してアクセスされる．オブジェクトハンドルは，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(注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`$\{\alpha\}$” 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 し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．

(注1)：Plan9 ソースコードを元に修正したので，プログラム上ではこのテーブルのタイプ名は Chan(nel) となっている

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(...)` 等) 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

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，対応したサーバントに

処理を行わせる．LP49core は §4.2 で述べたように，objH テーブル経由でサーバント内の抽象ファイルにアクセスする．サーバントを bind 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 (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．

分散処理は，次節 6. で説明する．

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．

(3) サーバマウントとマウントサーバント

クライアントは，サーバ登録簿から目的サーバを見つけて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．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] 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．マウントサーバントは、サーバ上のオブジェクトにアクセスするための **Proxy オブジェクト** を提供する．Proxy オブジェクトとは、以下の内容のオブジェクトハンドル (objH) である; (a)type フィールドはマウントサーバントを指す．(b)mchan フィールドは objH{ サーバリンク } を指す．(c) qid フィールドは サーバオブジェクトの qid 値.

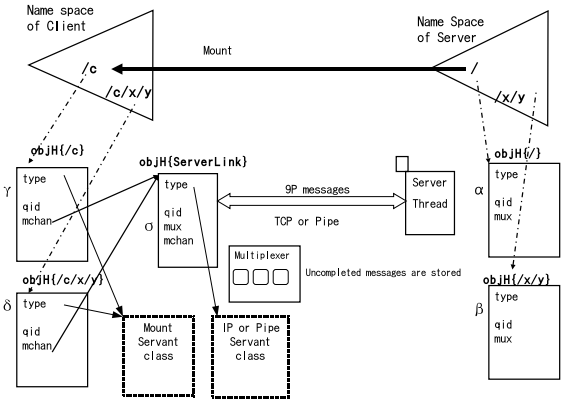


図 4　リモートオブジェクトアクセス
Fig. 4　Remote object access

Proxy オブジェクトに対する操作は, マウントサーバント (type フィールド) によって 9P メッセージに変換され，サーバリンク (mchan フィールド) を通してサーバに送られ，目的オブジェクト (qid フィールド) に対して処理が行われる．

サーバの開始点の Proxy オブジェクトはサーバをマウントした時に割り当てられる．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．こうしてサーバ上のオブジェクトもローカルオブジェクトと同様にアクセスされる．

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.

(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: 次の mount コマンドによりサーバをマウントする．ここに， /srv/dos はサーバリンク，/c はマウントポイント，/dev/sdC0/dos はサーバ名前空間のマウント開始位置，-ac は書き込み可能マウントを意味する．

mount -ac /srv/dos /c /dev/sdC0/dos

LP49core は，これによりサーバとメッセージのやり取りを行いサーバをマウントする．具体的には，マ

ウントポイントに “objH{/c}” (図の γ) テーブルを割り付けて，サーバ内の “/” オブジェクト (図の α) の **Proxy オブジェクト**として機能させる，このテーブルの代表的フィールドの値は次のとおりである: (a) type フィールド: マウントサーバント (#M). (b) mchan フィールド: objH{ サーバリンク }. (c) qid フィールド: サーバ内の開始位置オブジェクトの qid.

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.

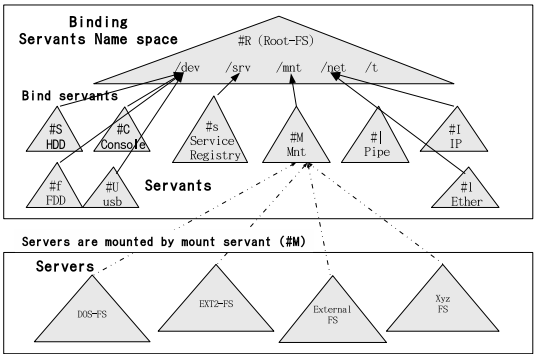


図 5　サーバ，サーバントと名前空間
Fig. 5　Server, servant and Name space

(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．LP49core 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る．このテーブルの代表的フィールドの値は次のとおりである: (a) type フィールド: マウントサーバント (#M). (b) mchan フィールド: objH{ サーバリンク }. (c) qid フィールド: サーバ内の/b/c/d オブジェクト (図の β) の qid．つまり，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.

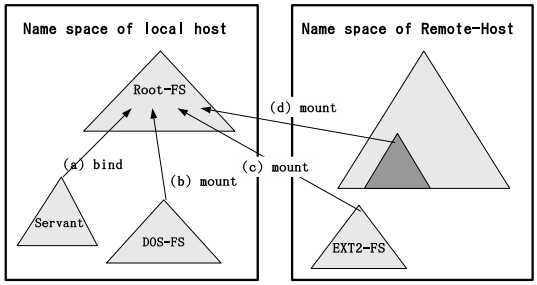
名前空間の構成法を図 6 に示す．図中の (a) では，指定されたマウントポイントにサーバントを結合 (bind) することにより，サーバントの名前空間がプロセスの名前空間に接続され，サーバントのサービスを受けられるようになる.

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 3 回の改良（Unix system V の Streams 型, x-kernel 型 [12], 現方式）を



(a) A servant is mounted onto my name space.
(b) A local server is mounted onto my name space.
(c) A remote server is mounted onto my name space.
(d) A sub name space of remote-host is mounted onto my name space.

図 6　名前空間とマウント
Fig. 6　Name space and mount

経ただけに適切にモジュール化されており，比較的軽い修正で LP49 に移植することができた．

インターネットサービスは，IP サーバント (“#I”) が提供している．IP サーバントの下で 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
  |           |- 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 [5] 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行える (単一ソース修正の場合，make して LP49 立ち上げまで 10 秒程度．全 make で 40 秒) ので，大変快適である．

8. 関 連 研 究

分散処理機能の観点からは，LP49 は Plan9 [1], [2] の延長にある開発・学習キットとも云える．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た．9P プロトコルの採用により，多くの Plan9 サーバを少ない修正で 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，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ので，LP49 では人手でプログラム修正(注2) を行い GNU 環境で開発できるようにした．

L4 [3], [4] 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をするこ

(注2)：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
Table 1　LP49 v.s. Plan9

分類	Plan 9	LP49
Micro kernel	No	Yes
プロセス スレッド	コルーチン, メモリ域共用プロセス	L4 Process L4 Thread
システムコール	Trap プロセスがカーネルモード化	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 言語 無名フィールド typedef, USED(), SET() 自動ライブラリリンク	GCC の C 言語
コンパイラ Utility	Plan9 独自 C コンパイラ, Linker, Assem, mk	GCC コンパイラ, gld, gas, gmake
Binary	a.out 形式	ELF 形式

となく活用できた.

Minix [6], [7] 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．モノリシック OS の方が効率が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．Linux の世界でも最近はユーザレベルファイルサーバが盛んに研究されている．Minix は分散 OS 機能は範囲外である.

SawMil [8] 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 [9] 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 を使った仮想マシンといえる.

GNU の **Hurd** は，歴史あるマイクロカーネル OS 開発である．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** [10] 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.

Microsoft reseach の SIngularity [13] は，安全言語 Sing#の機構でプロセス保護を行う研究 OS である．マルチサーバという点で，同じ方向をめざしている.

9. おわりに

LP49 の全ソースコードと資料は，WEB サイト http://research.nii.ac.jp/H2O/LP49 [11] にて公開している.

9.1　今後の展開予定

(1) 広域サービスバス

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，サーバントとサーバのマルチプレクサであり，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と言える．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 “広域サービスバス” として更に簡潔化，融通性強化できる．これ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非同期型メッセージもサポートすべく検討中である．SIP [14] プロトコルが VoIP サービスを実現するように，9P は分散 OS サービスのプロトコルとしての可能性を持っている.

(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組み込む予定である．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．保持メモリ域は，プロセス停止後も消去されないメモリセグメントであり，プロセスが指定した適切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 roll back を行える.

(3) 最適化

現 LP49 には，高性能化の仕組みはほとんど組み込んでない．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．特にメモリページ割付を担当する Pager の機能拡張を行い，ページキャッシュ等を実現する予定である.

(4) 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，LP49 では Identity-Based Encryption [15] を使用した認証方式の可能性を検討している．

9.2 OS 学習開発キットとして

(1) 学習用分散 OS として

コメントを含むソース規模は，HVM: 約 2 K 行，LP49core: 約 68K 行 (内 19K 行がプロトコルスタック)，ライブラリ: 約 19K 行，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

コンポーネント化されていること，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, Plan9 のサーバやドライバを容易に移植できることなどの利点がある．

現 LP49 の測定値は, 実機 (Pentium3 1GHz) の場合は以下の通りである．(a)RAM ファイル読み出しは, 16B: 21 μ sec, 1KB：26 μ sec, 4KB：39 μ sec, 8KB: 62 μ sec. (b) 別マシンのファイル読み出し（§6. 3 参照）は，16B:34msec. 1KB: 36msec, 4KB: 54msec, 8KB: 98msec.

QEMU 上で走らせた場合 (Pentium4 2.8GHz, Fedora8) は以下の通りである．(a)RAM ファイル読み出しは，16B:97 μ sec, 1KB:107 μ sec, 4KB:157 μ sec, 8KB:373 μ sec. (b) 別マシンのファイル読み出しは，16B:59ms. 1KB:66msec, 4KB:78msec, 8KB:99msec.

Buffer cache も page cache も未実装なので，書き込み時間も読み出し時間と殆ど同じである．まだ充分な最適化は行っていないので性能的には改善の余地があり，前述のスレッド優先度制御，page cache などの導入で性能は向上できる．

文　　献

[1] Rob Pike, et al. : *Plan 9 from Bell Labs*, Proc. of UKUUG (1991).

[2] Bell labs.: *Plan 9 Web site*, http://plan9.bell-labs.com/plan9/

[3] J. Lidtke: *On micro-Kernel Construction*, Proc. of IEEE International Workshop on Object-Orientation in Operating Systems (IWOOOS), (1996).

[4] Karlsruhe univ. : *L4 Web site*, http://l4ka.org/

[5] Qemu project : *Qemu Web site*, http://www.nongnu.org/qemu/

[6] A. Tannenbaum & A. Woodhull: *Operating systems design and implementation,3/E*, Addison-Weisley, (2000).

[7] www.minix3.org.: *Minix Web site*, http://www.minix3.org/

[8] A. Gefflaut, et al.: *The SawMill multiserver approach*, 9th SIGOPS Wuropean Workshop, 2000.

[9] DROPS OS project : *L4Linux Web site*, http://os.inf.tu-dresden.de/L4/LinuxOnL4/

[10] GNU Hurd project : *L4Hurd Web site*, http://www.gnu.org/software/hurd/history/port_to_l4.html

[11] *H_2O : LP49 Web site*, http://research.nii.ac.jp/H2O/LP49/

[12] N. C. Hutchinson & L. L. Peterson: *The x-Kernel: An architecture for implementing network protocols*, IEEE Transactions on Software Engineering, 17(1), Jan. 1991.

[13] G. Hunt. et. al.: *An overview of the Singularity Project*, Microsoft research technical report MSR-TR-2005-135, 2005.

[14] IETF : *RFC 3261: Sessin initial protocol*, http://tools.ietf.org/html/rfc3261

[15] Stanford univ. : *Pairing-based cryption library Web site*, http://crypto.stanford.edu/pbc/

（平成 xx 年 xx 月 xx 日受付）

丸山　勝巳　（正員：フェロー）

昭 45 東大大学院修士了．同年–平 7 電電公社 (NTT)．現在，国立情報学研究所教授．高水準言語，実時間システム，並行オブジェクト，分散 OS 等の研究開発に従事．昭 57 電電公社総裁表彰，平 5 情報処理学会論文賞，平 9 電気通信普及財団賞

佐藤　好秀

平 15 年日立製作所入社，平 18 日立工業専門学院研究科了，国立情報学研究所にて分散 OS 研究に従事，現在日立にて金融系 SE,

Abstract Control systems (e.g. embedded systems, home servers) are becoming more and more sophisticated, networked and complex. Software dependability and productivity are the first concern of their design, and Operating Systems play very important roles. It will be nice to have a distributed OS kit for studying and developing. LP49 is a component-oriented OS with micro-kernel and multi-server architecture, and suitable for this purpose.

Key words Distributed OS, L4, Plan9, Micro-kernel, Multi-server, Component