社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－OS－108 (17)
2008／4／24

連携処理のためのコンポーネント型ＯＳ LP49

丸山勝巳 *1　佐藤好秀 *1*2
*1:国立情報学研究所
*2:（株）日立製作所

組込みシステムを始めとする制御系プログラムには，強い耐障害性・連携処理の容易化・プログラム開発の容易化などが求められる．本報告では，多数のプロセッサが連携動作する組込みシステム向けのオペレーティングシステムとして LP49 を紹介する．LP49 は，L4 マイクロカーネルの先進的な機能・性能・融通性と Plan 9 の簡潔強力な分散処理機能・名前空間を活用し，コンポーネント化と連携処理を強化するものである．また制御プログラムに効果的な Hoare-CSP 的記述もサポートする．

キーワード　L4 マイクロカーネル，Plan 9，マイクロカーネル，組込みシステム，分散 OS

Component-oriented OS for federated embedded systems: LP49

Katsumi Maruyama *1　and Yoshihide Sato *1*2
*1: National Institute of Informatics
*2: Hitachi, Ltd.

Embedded systems software is required to have enhanced features such as federated processing, dependability and efficiency. Ease of program development is also very important. LP49 is a component-oriented OS with micro-kernel and multi-server organization to answer these requests. 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 will support easy development of federated embedded systems.

Key words　L4 micro-kernel, Plan 9, Embedded system, distributed OS

1　はじめに

組込みシステムを始めとする制御系プログラムは，機能の高度化，多数のプロセッサによる連携処理，高信頼化，プログラム開発期間の短縮，実時間処理など

が求められている．高機能化に対しては新しいプログラムモデル，高信頼化に対しては適切なモジュール化と障害閉じ込め，プログラム開発の容易化に対してはプログラムのユーザモード化，コンポーネント化が有効である．本報告では，多数のプロセッサが連携動作する組込みシステム向けのオペレーティングシステムとしてLP49を紹介する．LP49は，L4マイクロカーネル[1,2]の先進的な機能，性能，融通性と，Plan 9[3,4]の簡潔強力な分散処理機能，名前空間を活用し，コンポーネン化と連携処理を強化するものである．

なお，ホームサーバも分散組込みシステムと類似動作するため，本OSの適用範囲内と考えている．

2 本OSのねらい

2.1 組込みシステム向け達成目標

組込みシステム向けに，以下を狙っている．

- 組込みシステムの多種多様なデバイスのドライバ開発の容易化．
- 組込みシステムプログラムの保守性，拡張性向上と開発の容易化，短期間化．
- ソフトウェアバグは根絶できない．部分障害によるシステム停止の防止と，障害部のみの部分再開を出来るようにする．
- プログラム規模的にはソフト実時間処理が多く，ハード実時間処理は専担プロセッサも使われるので，前者をより重視．
- 分散ＯＳの開発や研究に使える部品を提供したい．

2.2 L4マイクロカーネルの活用

以下の観点から，マイクロカーネル＋マルチサーバ構成を採用している．

(1) マイクロカーネル以外のプログラム（デバイスドライバも含む）はユーザモードで走るので，プログラム開発，デバッグが圧倒的に容易化できる．
(2) 障害が生じても，そのプロセスだけを停止，再開させることが容易にできる．
(3) コンポーネント化に適する．
(4) メッセージ結合なので，分散処理の実現が容易．
(5) L4マイクロカーネルは，我々が要求する機能を簡潔，効率的に実現している．またメモリ管理もプログラムでき融通性に富むOSが可能である．

2.3 マルチサーバ構成

ある部分で障害が生じた場合，他の部分への悪影響を最小にして，部分再立ち上げが出来るべきである．そこでサービス毎に独立のユーザモードプロセスを設け，障害プロセスのみの再起動を可能にする．

L4は割込みをメッセージ化する機能がある．デバイスドライバはバグが残存しやすいが，ユーザモードプロセスの中に置くことにより，耐障害性の面でも非常に有利である．大部分のドライバは，応答時間の微小増よりも耐障害性強化が重要である．

2.4 コンポーネント化

大小2種類のコンポーネント化を進めている．サーバプロセスは，パイプや通信コネクションでつなぐだけで，後述の9Pプロトコルを喋って仕事を行ってくれる自立的なコンポーネントである．もう一つは，デバイスなどに対応した小粒なコンポーネントである．

2.5 連携処理の容易化

最近のシステムは多数のプロセッサによる連携処理が要求され，(middleware でなく)OSによる融通性に富む連携処理のサポートが重要である．

リモートリソース管理に関して，Plan 9はプロセス個別名前空間と名前空間のマウントという簡潔で強力な手法の実現により，リモート名前空間を自分の名前空間にマウントするだけで，local/remote を意識せずに使えるようにしている．本システムでは，この手法を継承することとした．

2.6 9Pプロトコルと Plan9 ソース活用

連携処理プロトコルは，専用設計すれば機能も性能も最適化できるが，普及させるには標準プロトコルが望ましい．実績あるプロトコルのうち，分散ファイルシステムプロトコル(Ex. NSF, ,,)では機能不足であるが，Plan9 の 9P プロトコルは必要な機能を持つので，これを採用した．また Plan9 はオープンソース化されたので，できるかぎりソースコードも流用することとした．なお，今後は RPC(remote procedure call)によらない非同期メッセージ形プロトコルも発展すると予想している．

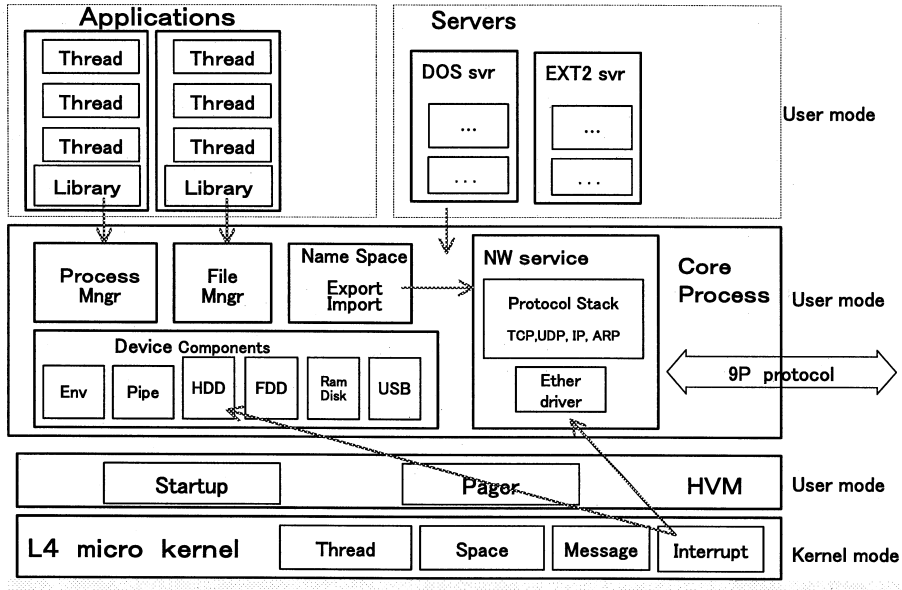


図 1 構成概要

2.7 CSP 的プログラム記述のサポート

制御プログラムでは，多数のリソースが並行動作しており，メッセージを受けたり制御したりする．つまり並行処理が本質であり，L4 の軽量スレッドと効率的で融通性に富むメッセージが効果を発揮できる．制御プログラムの重要な処理パターンに，複数のメッセージの選択待ちがある．UNIX 系 OS では select 文が用意されていが，記述性も効率も悪い．そこで，Hoare CSP[5]の Guarded command などを使えるようにする．

3 OS の基本構成

3.1 基本構成

図 1 に LP49 の構成を示す．

(1) L4 マイクロカーネル

スレッド制御，メモリ空間制御，メッセージ機能を提供している．唯一カーネルモードで走る．

(2) HVM プロセス

システム起動と pager を含む．Pager は，各スレッドがページフォールトが生じた場合に適切なページを割り当てる機能を持つ．Page Fault の措置をユーザモードで記述できることも L4 の特長のひとつである．

本プログラムは次のコアプロセスと一体化できるが，将来仮想マシン化への発展を考えて Hypervisor Monitor として独立プロセス化した．

(3) コアプロセス

Plan 9 ではカーネルが提供している機能の大部分をユーザモードのコアプロセスが実装している．

(a) プロセス制御: L4 機能を使ってプロセス生成・消滅などを行う．

(b) ファイルアクセス: ファイル関連のシステムコールを受けて，名前空間を調べ，該当サーバに処理させる．

(c) 名前空間管理: 名前空間のマウントなどを管理し，パス名から目的リソースへのアクセスを可能にする．

(d) デバイス制御: デバイスドライバは，本プロセスに含まれる．

(e) ネットワークサービス

(4) サーバ群

DOS ファイルサーバ，Ext2 ファイルサーバなど,各サービス毎にプロセスを用意している．9P プロトコルを喋れる普通のユーザモードプロセスにすぎない．

(5) 応用プロセス

CSP 的プログラミングも支援する．

3.2 Plan 9 -> LP49 の内容

各種便宜関数を用意することにより，Plan 9 のオー

プンソースを出来るだけ活用できるようにした.

(1) プロセス・スレッド・排他制御: 頭脳作業

Plan 9では, プロセスはコルーチン, スレッドはメモリ域を共有するプロセスとして実装されている. LP49 では L4 のスレッドと空間管理で全面的に置き換えた. 排他制御に関しては, L4 はメッセージで実現する方針のため semaphore 等の低レベル機能は備えていないが, Plan 9 のソースを流用するために lock() など幾つかの低レベル関数を用意した.

	Plan 9	LP49	質	量
名前空間	簡潔で強力	Plan9 を踏襲		
9Pプロトコル	実績がある	9P を踏襲		
メモリ域管理		L4 Pagerを利用	難	中
プロセス	コルーチン+占有メモリ空間	L4スレッド+スペース	中	中
スレッド	メモリ空間を共用したプロセス	L4スレッド	中	少
排他制御	多彩なlock機構	Primitive追加	中	多
システムコール	Trap	L4メッセージ	中	少
システムコール引数引継	カーネルが直接ユーザ空間をアクセス	L4メッセージの値引継・バッファコピー・ページマップ	難	少
OS並行サービス	ユーザプロセスが変身	マルチスレッドサーバ	難	少
デバイスドライバ	カーネルモード	ユーザモード	中	中
低レベル記述		いくつかの関数を定義	中	中
例外処理	Longjmp的技巧	関数値で報告	中	多
各サーバ	個別ユーザプロセス	個別ユーザプロセス	易	中
コンポーネント化	その"きざし"	整理して改善を図る予定。		
言語仕様	独自C(無名フィールドなど)	GCC	易	多

表 1 Plan 9 と LP49 の対比

(2) システムコールとライブラリ: 修正量大

Plan 9 のシステムコールはトラップ方式であり, ユーザプロセスロがカーネルモードになって処理される. LP49 では, システムコールはライブラリでコアプロセス宛 L4 メッセージになる. サービス中に中断があり得るので, マルチスレッドサーバーを実装した.

(3) メモリ管理: 修正箇所は少ないが頭脳作業

LP49 のスレッドは Pager を登録できる. スレッド実行中に page fault が生じると, 登録された Pager に page fault メッセージが送られる. そこで Pager は適切なページを割り当てるのである. 現在は, HVM プロセスの Pager を登録しているが, ユ.ーザが自前の Pager を定義してそれを登録することも可能である. 現 Pager は基本機能のみ用意しているが, 適用分野の要求に応じて工夫することにより, 特別なメモリ管理を組み込むことが可能である. L4 はまた IO 空間のマッピングの機能も有している.

(4) 低レベルプリミティブズ: X86 と PC アーキテクチャに合わせてひたすら作業.

(5) デバイスドライバ: 比較的簡単に移植可能

デバイスドライバの開発は面倒なので, Plan9 のドライバをできるだけ小幅な修正で使えるように, 周辺機能を整えた. 割込みハンドラは, 該当割込みが生じた時の通知先を L4 カーネルに教えておくことで, 割込みが生じるとメッセージが届きハンドラが起動される. ドライバプログラムに於ける修正は, 殆ど物理メモリ番地アクセスに伴うものであった.

(6) 例外処理: 単純だが修正箇所多い

Plan 9 は setjmp/longjmp に類似したスタック操作手法で throw/catch 形の例外処理を実現しているが, これはマルチスレッド環境では効率的でない. そこで, 関数戻り値でエラーを引き継ぐ方法に変更した.

(7) C 言語仕様の差: 単純だが修正箇所多

Plan 9 の C は, 構造体に無名フィールドを許す等, 独特な方言である. LP49 は使い慣れた GCC を採用した. Plan9C->GCC 自動変換ツール作成も考えたが run once なので手動変換した.

3.3 システムコールの動き

図 2 にシステムコールの動作を示す. モノリシック OS とは異なり, LP49 のシステムコールはプロセス間メッセージ通信とマルチスレッドサーバで実行される. L4 メッセージの値引継, バイト列コピー, ページマップ能力を使い分けて効率をあげている.

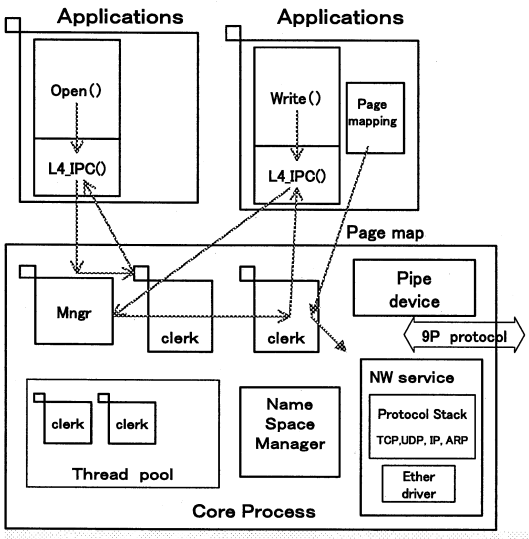


図 2 システムコールの動作

(a) マルチスレッド構成: システムコールは, ライブ

ラリで L4 メッセージに変換されて mngr スレッドに送られる．Mngr スレッドはスレッドプールから clerk スレッドを割り当てて，システムコールの実務を担当させる．

(b) “open(パス名，モード)” の例: パス名引数はバイト列コピー，モード引数は値引継にてクライアントプロセスからコアプロセスに渡される．Clerk スレッドは，Name Space Mngr を用いて対象を同定して open 操作を行う．Name Space Mngr は，名前空間のマウント内容に応じて，場合によっては複数の遠隔ファイルサーバに問い合わせを行ない，対象を同定する．

(c) “write(fd, buf, n)”の例: クライアントプロセスの buf 域を含むページをコアプロセス空間にマップして，コピーを避けている．ファイルサーバがローカルの場合はパイプを，リモートの場合は通信接続を介して 9P プロトコルで交信する．

4 名前空間と連携処理

Plan9 の以下の優れた機構は，LP49 でも踏襲している．

・全てのリソースをファイルトリーとして表す
・各プロセスが個別に名前空間を持てる
・リモートリソースであっても，その名前空間を自分の名前空間にマウントすることにより，ローカルとまったく同様にアクセス可能になる．
・９Ｐプロトコルによるサーバインタフェース

本章では LP49 の連携処理に関する話題を説明する．

4.1 名前空間

リモートマシン 192.168.1.2 のサービス(ポート番号で指定．ここでは 456 とする)を使うために，自分の名前空間にマウントするプログラム例を考えよう．

```
fd = dial(“tcp!192.168.1.2!456”);
//リモートサーバ 456 に TCP 接続する．
mount(fd, -1, “/m/n”, flag, “”);
//目的サーバのルート””を, ”/m/n”にマウントする
```

これにより，リモートサービスは”/m/n”にマウントされ，”/m/n/e/f” などへのアクセスは，リモートサーバで処理されるようになる．

4.2 リモートアクセスの仕組み

図 3 は，Plan 9 及び LP49 のリモートリソースアクセスの説明である．Chan(nel)構造体は，file descriptor が指すテーブルであり，目的リソースをアクセスするためのテーブルである(Linux で云えば VFS の file 構造体に近い)．図にはローカルリソースアクセスの例も描いておいた．

(1) リモートサーバに接続するには dial( ) を実行する．これによりサーバとの接続が行われ，“サーバ接続 Chan”(図の Chan: server)が用意される．

(2) Mount( )を実行して，サーバを自名前空間にマウントすると，“サーバ接続 Chan”にリンクした”マウント Chan”(図の Chan: /m/n)が割りつけられる．“マウント Chan”は，与えられたパラメータから 9P メッセージを編集し，“サーバ接続 Chan”に渡して，リモートプロシージャコールを実行する．

(3) S1 のファイル/m/n/e/f を open( )すると，サーバと 9P プロトコルをやり取りして該当 Chan 構造体（図の Chan:/m/n/e/f）が作られる．

(4) read( ) するとサーバと 9P プロトコルをやり取りしてデータを読み出す．

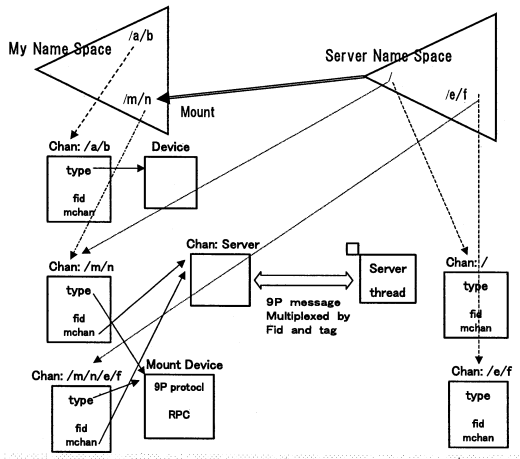


図 3 リモートリソースの操作

4.3 名前空間の export と import

ファイルサーバの名前空間を自分の名前空間にマウントするだけではなく，自分の名前空間の全体あるいは部分を別マシンに見えるようにすることも可能である．これは名前空間の export, import と呼び，Plan 9 を継承している．例えば，端末 PC で自分の名前空間

を export し, CPU サーバがそれを import することにより, 端末 PC の環境(名前空間, 環境変数など)を引き継いだまま CPU サーバ上で処理を行える. 端末 PC が CPU サーバの強力な計算パワーを手に入れたようなものであり, rlogin よりも格段に便利に使える.

5 コンポーネントの観点

図 4 は, LP49 が目指しているコンポーネントモデルである. このモデルは大小 2 種類のコンポーネントを意味している.

5.1 プロセスレベルの部品

サーバは, 9P プロトコルを喋る自立したユーザモードプロセスである. クライアントとは, パイプ(同一ノード内の場合), TCP コネクション(別ノードの場合)を介して会話する最も素直なコンポーネントである.

制御システム用としては, イベントの登録通知などを 9P プロトコルに追加する必要となるかもしれない.

5.2 プロセス内の部品

デバイスドライバなどのコンポーネント. これには, 次の二種類がある.

(1) **受動コンポーネント**: スレッドを持たずに関数呼び出しで使われるもの.

(2) **能動コンポーネント**: 自前のスレッドをもち, メッセージで交信する.

Plan 9 のカーネルには, 能動コンポーネントに該当するものは無いが, LP49 は L4 スレッドという軽量強力な機能があるので能動コンポーネント化することにより, より自立性が高まる.

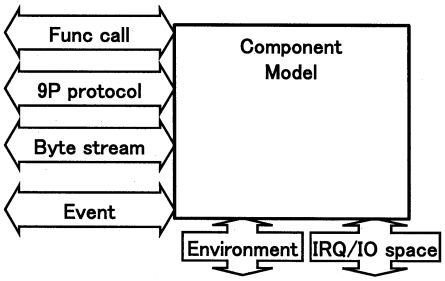


図 4 コンポーネントインタフェース

Plan9 のデバイスドライバ類は, 呼び出し方は標準化されているが, その先は絡み合っている. 現時点の LP49 は, まだコンポーネント化には手が回っていないが, 機能実装が終了した段階でコンポーネント化の整理を行う予定である.

6 CSP 的 APL プログラム

6.1 制御プログラムと CSP モデルの有効性

組込みシステムプログラムでは, 多数のリソースの並行管理・並行制御が求められことが多く, マルチスレッドが効果を発揮する. またプログラムパターンでいうと, 図 5 に示すようにある状態でいくつかのイベントを待っていて, イベントが到着するとそれに応じた処理を行って, 次の状態に遷移するという状態遷移モデルが頻繁に出てくる.

並行イベント待ちは, Unix 流 OS では select( )コールあるいは pole( )コールで記述できるが, 効率も記述性も非常に悪い. これに対し, Hoare の CSP モデルの以下の機能は, 大変魅力的である.

- Guarded command [論理式->コマンド []...]
- Parallel command コマンド || コマンド
- 入出力コマンド プロセス?変数 プロセス!式
- Pattern matching プロセス?P( ) プロセス!V( )

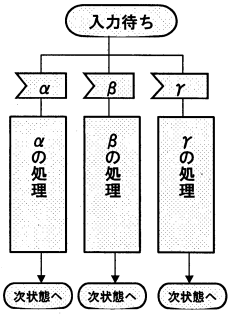


図 5 並行イベント待ち

幸いなことに, L4 スレッドは CSP モデルとの類似性が高い.

- L4 メッセージは同期通信である.
- L4 メッセージは, スレッド宛に送られる
- 複数ソースからメッセージを待つことができ, 受信メッセージに応じて該当処理を行うことができる.
- スレッドが軽量なので多数スレッドを実動できる.

そこで, APL プログラムでは L4 スレッドを使った CSP 流プログラムを可能にするために, 以下の検討を

進めている.

6.2 並行イベント待ち

並行イベント待ちは, L4 では以下のような感じで簡潔に記述できる.

```
L4_MsgTag_t tag;
L4_Msg_t  msg;
L4_ThreadId_t  client;
 tag = L4_Wait(&client); //receive from anyone
 L4_Store(tag, &msg);
 "tag や msg の内容をみて要求処理を決定する.
 必要ならば現 msg をストアして次 msg を分析する"
 switch(…) {
 case alpha: "Alpha の場合の処理"
              goto 次状態;
 case beta:  "Beta の場合の処理"
          ::
 }
```

L4 メッセージは, 値引継の他にバッファ間コピー, ページマップが行える. ただし, リモートノード間ではページマップの代りにページコピーとする. バッファ間コピーやページマップを行う場合には, 送信側・受信側ともに領域記述子を用意する. L4 メッセージの先頭ワードには値引継の個数, 領域記述子の個数とラベル(ユーザデータ)が載っており, 受信側ではこの先頭ワードから適切な処理を選択できる.

6.3 イベント記述子とその引継

Plan 9/LP49 では, ネットワーク接続も含めてリソースはファイルとして抽象化されているが, 例えばパケット受信等を read( )でなく並列イベント待ち文脈で扱いたい場合が多い. そこで ready event 記述子という機構を考えている. Ready event 記述子は, 以下の内容を持つ.

- 無中断で即読み出せるデータが準備できたことを意味する.
- スレッドは, 担当サーバに ready event メッセージを送ってくれるように登録をしておく. これにより, サーバは ready になる毎に ready event メッセージを講読スレッドに送る.
- ready event 記述子は, 別スレッドに引継ぐことも可能である.

図 6 は, ready event の引継の例である. LP49 ではネットワーク接続もファイルであるが, パケット到着は read()によらずに ready event メッセージとして扱うことが出来る. また, ready event 記述子を受信したスレッドは, データの先頭だけを読んで別の担当スレッドにイベント記述子を受け継ぐことができる.

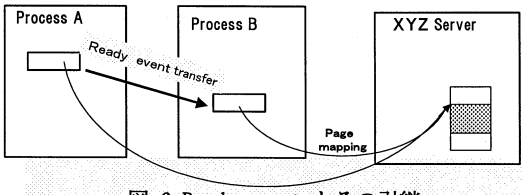


図 6 Ready event とその引継

6.4 CSP モデルの分散環境でのサポート

CSP モデルを分散環境で使えるようにするために, 図 7 に示すように L4 メッセージ(CSP メッセージ)を 9P プロトコルに載せて運ぶ.

CSP は同期通信であるが, 分散処理においては非同期通信を選択したい場合が多々あるので, 非同期も可能とする拡張を検討していく.

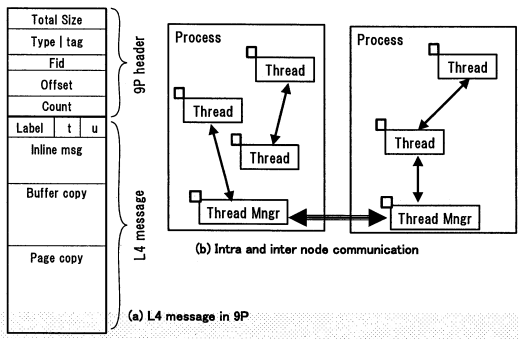


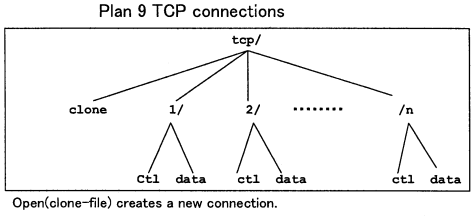
図 7 L4 メッセージと 9, プロトコル

6.5 リソース識別と名前空間

APL プログラムでは L4 スレッドを裸のまま使うことも可能であるが, 名前空間を経由してファイルとして見せることも可能とする. LP49/Plan 9 では, 全てをファイルとして抽象化するからである.

Plan9 の通信接続の扱いに合わせて, clone ファイルを open するとスレッドが生成され名前空間に登録される事とする. つまり図 8 に示すように, 目的スレッドを含むプロセスの名前空間は csp/clone ファイルを持つ. csp/clone ファイルを open するとスレッドが生

成され，名前空間には順次 csp/0, csp/1,,,として登録される．その下には csp/*/data, csp/*/ctl などのファイルがある．csp/*/data ファイルを write/read することで，メッセージが送受信される．



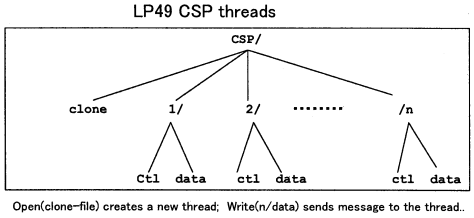


図 8 スレッドと名前空間

7 関連研究

- Plan 9: Unix の後継として Bell 研究所で研究開発．唯一実用化に至った分散 OS．優れた内容だけに普及しないのが残念．
- L4: ドイツの Karlsruhe 大学で研究開発．巧みな設計により，融通性の高い機能を効率よく実現した第 2 世代マイクロカーネル．
- Plan B: スペインの Rey Juan Carlos 大学で研究開発．Plan 9 ベース OS．Plan 9 はリソースをファイルに抽象化するが，Plan B は BOX 呼ばれる集合に抽象化し，リモートに公開する．
- QNX: 商用の POSIX 準拠の組込み向けリアルタイム・マイクロカーネル OS．
- L4-Hurd: GNU Hurd は，当初 Mach マイクロカーネルを用いたが，L4 に移植を始めた．

8 まとめ

組込みシステム用 OS には，必要な機能のみを組み込め，障害に強く，連携処理に強く，プログラム開発が容易なことが望まれる．それにはマイクロカーネル＋マルチサーバ構成 OS が有効なことを LP49 試作により説明した．マイクロカーネルとしては必要機能を効率よく簡潔に実現している L4 を採用した．マルチサーバ機構は，分散処理，9P プロトコル，個別名前空間，ユーザモードサーバなどで融通性に富む Plan 9 をベースに開発している．

現在，ファイルサーバは DOS, Ext2, Ramfs が、ドライバは HDD, FDD, Lance ether, USB1.1 等が動き始めている(図 1)．Plan 9 は高度な機能を簡潔に実現しているが，プログラム論理は難解で LP49 への移植は予定以上に時間を費やした．プログラム走行試験を VMware 上で行っていることと，L4 向け最適化が未着手なことから，正確な性能評価はまだ行っていないが，遅いという感触はない．

今後，機能面の増強(連携処理，CSP 的機構，各種サーバなど)，L4 向け最適化，実マシン上での性能評価などを行うとともに，ソースコードの整理とコンポーネント化を進め，マイクロカーネルとプロトコルスタックが揃った基盤ソフト学習素材としても利用してもらえることも考えている．

9 参考文献

[1] Jochen Liedtke: “On micro-kernel construction” SOSP 1995: 237-250

[2] L4ka web ページ http://l4ka.org/

[3] Rob Pike and Dave Presotto and Ken Thompson and Howard Trickey: ”Plan 9 from Bell Labs” Proceedings of the Summer 1990 UKUUG Conference, pp. 1--9, London, 1990.

[4] Plan 9 web ページ http://plan9.bell-labs.com/plan9/

[5] C.A.R.Hoare: “Communicating Sequential Processes”, Comm. of ACM. Vol.21, No.8. 1978.

[6] 谷口 秀夫, 乃村 能成, 他, “適応性と堅牢性をあわせもつ AnT オペレーティングシステム” 情報処理学会研究会報告, vol.2006-OS-103, pp.71-78 (2006)

[7] 佐藤, 丸山: “組込みシステム向けの分散・コンポーネント指向 OS の設計と開発”, 情処学会研究会報告, vol.2007-OS-104 (2007)

[8] LP49 web ページ (ソースコード含む) http://research.nii.ac.jp/H2O/LP49/index.html