Vol.2009-OS-112 No.4
2009/8/5

Plan9を用いた分散組込みシステムのプログラミングシステム

盛合 智紀[†1]　並木 美太郎[†1]

本研究では，分散システムのノードとして組込みシステムを用い，位置透過性の高い分散システムを構築する手法を提案する．分散システムの通信プロトコルにTCP/IPを利用し，Bell研究所が開発したPlan9と，ネットワークファイルプロトコルである9Pを用いることで，ネットワークを意識せずに，ノードの手続きをローカル空間でのファイル入出力のように扱うことで透過性とアクセスのしやすさを実現する．また，提案した分散システムを実現するために，数KB程度のRAMで動作するスタックマシン型のVirttual MachineであるS91と，C言語のサブセット的な開発言語C–’を用いた処理系を提供した．今回実装したシステムはCPUとしてColdFire MCF52233を用いた．その結果，ROMサイズは25KB，RAMサイズは15KB程度となり，実行時間については，処理内容にもよるが，ネイティブの375倍，インタプリタのSilentCと比べて1/10程度であり，ノードのプログラムを実行するために十分な実行時間である．

Programming System for Distributed Embedded System using Plan9

Tomoki Moriai[†1] and Mitaro Namiki[†1]

This paper describes programming system for distributed embedded system with location transparency using TCP/IP networkd and 9P protocol which is desined for Plan9 orepatin system. This system supports Remote Procedure Call interface with file I/O interface of Plan9. A prototype system includes embedded server implemented with Virtual Machine S91 for C–’ subset language of C. As the result of prototyping, using size ROM size is about 25KB, RAM size is about 15KB. In the compared execution time of native code and SilentC interpreter, S91VM marked about 375 times slower than native code and S91VM marked about 10 faster than SilnetC.

1. はじめに

本研究では，分散システムにおける組込み機器の資源を，Plan9[1)]のネットワークファイルシステムの通信プロトコルである，9Pプロトコル[2)]を用いて透過的に管理する，分散システムの構築法の提案と，提案手法のための，センサーノードのような少ない資源しか有さない環境下でも動くVM(Virttual Machine)と，言語処理系の設計と試作を行った．

1.1 背　景

近年，光ファイバーやADSLといった物に代表される情報インフラの拡充によって，何処でも高速なネットワークを利用することができるようになりつつある．さらに，近年の組込み機器の高性能化，低価格化も注目を浴びている．これは，CPUやメモリのみならず，入出力装置等の周辺装置に関してもあてはまることであり，その結果として，小さな組込み機器の上で，多種多様な入出力装置を利用できるようになった．

このような，拡充されたネットワークと，高性能化した組込み機器の一つの利用例として，たとえばセンサーネットワークが有り，地震検知や気温・湿度計測，プラントにおける在庫管理など，設置環境を把握する場面で活用できる．そのため，現在ではセンサーネットワークを構築するための小型組込み機器として，用途毎に様々な物が開発され，今後利用される機会が増える．

このように，ハードウェアの整備が進むにつれ，センサーネットワークを構築するためのシステムの開発も求められている．センサーネットワークをはじめとした，ネットワークと組込み機器の連携を実現するための一つの手法として，分散システムが挙げられる．

組込み機器を利用した分散システムを構築する際に，次の二つの課題が残る．一つ目が，分散システムの資源管理に関してである．二つ目が，センサーノードで行う処理を記述する際の，組込みソフト開発に関する問題である．

(1) 分散システムの資源管理

古典的な分散システムは，OSI参照モデルで言う所の4層目に当たるトランスポート層の通信プロトコルであるTCPやUDPをコード中で明示的に利用し，ネットワーク通信の利用を意識したプログラミングが求められるため，分散システムの導入の敷居を高めていた．最近では，ミドルウェアを利用し，分散システムを構築する際の通信プロトコルスタッ

†1 東京農工大学
Tokyo University of Agriculture and Technology

クとして OSI 参照モデルの 5 層以上に当たる IIOP[3] 等のプロトコルを提供し，プログラマに細かな通信内容を意識しない環境が提供され始めた．

しかし，ミドルウェアを利用するには，ミドルウェアの機能を提供するライブラリが開発環境に応じて存在する必要がある．また，ミドルウェアを利用してもネットワークを意識したコネクションの確立や，データの送受信待ちは明示的に示す必要がある．これは，プログラマにネットワークを意識させることとなり，分散システムを構築する際で重要な要素である，ユーザーにリソースの存在場所を意識させない分散透過性の視点から見ると，まだ改良の余地がある．

(2) 組込み機器の開発と保守

組込み機器に対するシステム開発では，各ベンダーが開発した各機器やサーバのアーキテクチャに応じた開発環境を揃える必要があり，アーキテクチャの種類が豊富にある分，様々な開発環境も求められることとなる．アーキテクチャの違いに対応するには，アーキテクチャ毎にソースコードを変更するか，各アーキテクチャ毎に VM を用意し，開発者は VM のソースコードを記述するかのいずれかである．分散システムは異なるアーキテクチャが前提となるので，システム中のノードのハードウェア構成がすべて同じであるとは限らず，物理的に離れた所にあるそれぞれのアーキテクチャを正しく把握した上で，ソースコードを変更するより，VM を利用した方が，生産性が高い．さらに，VM を利用することで，ネットワーク越しにノード上の手続きを書き換える作業も，行いやすいと言った利点が生じる．

2. 先行研究

センサーネットワークないしは，分散システムに関する先行研究を示す．

(1) LiteOS[4]

センサーネットワークを構築するノードを，ユーザーから見て，ファイルツリーの形式で管理できることが特徴である．管理のために，LiteShell と呼ばれる専用のツールを使うことで，UNIX ライクなコマンドによる操作を用い，ファイルツリーを移動して，ノードの資源を管理できる．しかし，MicaZ や IRIS[5] といった，センサーネットワークを構築する際に，多く用いられるハードウェアを対象とした OS である．センサーネットワークを利用するには，サポートされたハードウェアを PC に接続し，接続したハードウェアとノードにそれぞれ専用の LiteOS を載せる必要がある．ハードウェアへの依存度が高いことと，専用のネットワークの利用が必要となり改善の余地がある．

(2) Inferno[6]

Bell 研究所が Plan 9を元に作成した分散 OS である．設計理念や通信プロトコルを含む大部分は Plan9 と共通である．Plan9 との最大の違いは，Inferno が独自に開発した Dis と呼ばれる VM 上で動作していることであり，Dis の機械語を生成する為に，Limbo と呼ばれるプログラミング言語を用いた開発環境を用いる．Dis は Windows や Linux 等をホスト OS として，アプリケーションとして動作させることも，x86 は勿論，ARM や PPC アーキテクチャ上で直接動作させることもできる．しかし，Dis はメモリ管理ユニットの無いハードウェアでも動作するとされているが，1MB 程度の RAM は必要になってしまう．

(3) LP49[7]

マイクロカーネルである L4Linux と，Bell 研究所によって UNIX の次の OS として開発した Plan9 を組み合わせた分散 OS．Plan9 の 9P プロトコルを用いることで，すべての資源をファイルに仮想化して管理することができる点と，L4 マイクロカーネルによる信頼性・頑強性・維持管理性の提供にある．L4 の過去の資源を活用できることも利点の一つである．しかし，L4 は GCC を用いて開発できる反面，ネイティブコードを生成して利用するため，ノードのアーキテクチャを把握する必要が生じると同時に，OS のサイズが大きいことが問題である．

3. 目　　的

本研究では分散システムの通信プロトコルに TCP/IP を利用した 9P プロトコルを用いることで，ノードの手続きをローカル空間でのファイル入出力のように扱う，位置透過性の高い分散システムの構築法を提案する．これにより，ソケットの生成など，ネットワークを意識する必要が無くなり，RPC(Remote Procedure Call) の実行をファイル入出力のみで行える．また，提案した分散システムを実現するために，数 KB 程度の RAM で動作するスタックマシン型の VM と，C 言語のサブセット的な開発言語を用いた処理系を提供することを目的とする．

この方式により，クライアントに当たる PC は，9P プロトコルさえ利用できれば，他に必要なツール等は無く，本研究によって提案するノードを利用できることを意味しており，ノード上の手続きをファイルに仮想化することで，ファイルの読み込みが RPC に該当することが，本方式の特徴である．つまり，クライアント/サーバ間の通信のみならず，資源管理においてファイル入出力を用いる単一的なモデルを提供できる．

さらに，VM を利用することで，アーキテクチャ依存部分以外のコードを使いまわすこと

がし易い．また，ノード上の手続きをネットワーク越しに変更することも容易になる．物理的に離れた所に点在しうる，分散システムの利用を考えた際に，有用な機能である．

4. 設　　計

以下において，クライアントは RPC を呼び出す側を表す．一方で，サーバはクライアントからの RPC の要求を処理する側で，サーバは本研究において作成される，独自の処理系を載せたノードを指す．本研究では，クライアントに要求する要件は 2 点で，TCP/IP 通信のサポートと，9P プロトコルを理解するための機構である．9P プロトコルを処理できれば，PC がクライアントである必要は無く，クライアント側に手を加えずに，サーバの資源を利用できることが特徴である．

また，サーバ上で動作させる，S91 と呼ばれるスタックマシン型の VM を作成し，利用する．この S91VM 用の実行コードを出力するコンパイラとして，C 言語ライクなシンタックスを持つ C–’ 言語という物を用いる．C–’ は C 言語のサブセットに，S91VM において RPC の対象として，ファイルに仮想化してクライアントに見せる手続きを選別するための修飾子等を追加した仕様となっている．S91VM がサーバ上では 9P プロトコルスタックを保持しており，通信内容を判断して，適切な処理を行う．

4.1 システムの構成図

システムの全体図を示す．図 1 では簡略化のため，サーバの保持するファイルツリーを省略している．サーバの保持するファイルツリーの構造については，4.3 章で述べる．

図 1 において，クライアントのファイルシステムの部分木として，サーバの VM に登録されている手続きが含まれる．サーバの提供するファイル木は，サーバ上の VM で動作する命令コードから生成されたものであり，VM の実行を管理する物や，RPC の結果を取得する，オブジェクトの階層になっている．そこで，上記のファイルとして仮想化するための情報として，関数名や変数名が別途必要になるので，サーバで動作させるコードファイルに，必要な情報を埋め込むことで，サーバ上で関数名の設定等の処理を行う必要がないことも，本方式の特徴である．

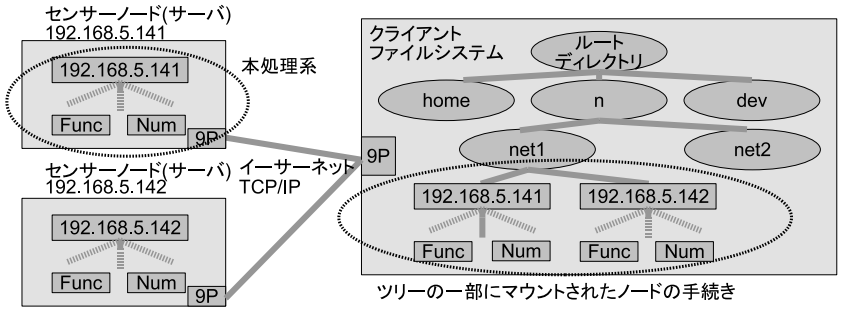

図 1　システム構成図

4.2 ノードの提供する機能

ノードはクライアントからの RPC の要求を受け取り，内容に応じた処理の結果をクライアントに送り返す．RPC を実行する際のインターフェースにファイル入出力を用いており，クライアントはノード上の RPC の対象となる手続きを仮想化したファイルを読み込むことで，RPC の実行を指示することと，結果を取得することを同時にできる．従来手法であれば，RPC の呼び出しを支持し，値が返ってくるまで待つ命令を明記する必要が有った内容が，ファイルの読み込み一つに集約される．

ノードの提供する機能としては以下の物がある．

- ノード上で動作するコードを，クライアントから取得し，ロードする
- ロードしたコードを管理する為のインターフェースをファイルに仮想化して，クライアントに提供する
- 複数のコードを同時に実行する

ノードの提供する機能を利用するために，ノード上で操作するコードをロードする方法と，クライアントからノードの内の資源を利用する方法について述べる．

4.2.1 ノードの処理内容の記述方法

ノード内の処理を記述するには，C–’ 言語を用いてコーディングを行う．C–’ によって記されたコードはコンパイラによって S91 バイトコードにコンパイルされる．C–’ は C 言語のサブセットではあるが，独自に拡張したシンタックスとして，RPC の対象とする関数と，それ以外の関数の扱いを区別する為の識別子である setrpc がある．setrpc が記述されている手続きは，コンパイラによって RPC の対象となる手続きと解釈される．RPC の対象となる手続きは，クライアントからノードを見るときに，ファイルに仮想化され，ファイル名等の必要な情報が S91 バイトコードに埋め込まれる．また，RPC の実行依頼を処理している時間外で動作する main 関数も存在する．S91 バイトコード内で RPC 対象の手続きとグローバル変数は，クライアントから利用される為のインターフェースとしてファイルに仮想化され，member ディレクトリに格納されている．

ただし，ホストと通信する為の pipe ファイルや，他ノードの RPC を実行する為の IP

GetTemp.s91

```
int a;                  /* グローバル変数 */
int final b = 1;        /* 書き換え不可能 */

setrpc int Get() {  /* RPC対象関数 */
      int temp;

      /* sence_adの呼び出し */
      temp = sence_ad();

      /* RPCの結果として返す */
      return temp;
}
```

```
int sence_ad() {   /* RPC対象外関数 */
      ・・・
}

/* 引数付きRPC対象関数 */
setrpc int GetArg(int num) {
      ・・・
}

void main() {
      while(1){・・・}/* RPC実行時以外の処理 */
}
```

図 2 C–’ で記述されたノードのソースコードの例

表 1 C–’ 用 API の設計

名前	機能
mount(ip,*dir_fp)	他のノードをマウントする
umount(*fp)	他のノードをウンマウントする
fopen(*fname,type)	通信用ファイルを開く
fclose(*fp)	通信用ファイルを閉じる
fgetc(*buf)	通信先から値を取得する
fputc(c)	通信先に値を送る

```
Linux(Plan9)

modprobe 9p

～ノードのマウント～
mount -t 9p -o proto=tcp
192.168.5.141 /n/node/

～VMテンプレートの継承～
mkdir /n/node/192.168.5.141/vm/Ins_GetTemp

～インスタンス化～
echo new > /n/node/192.168.5.141/vm/Ins_GetTemp/ctrl
cat GetTemp.s91 > /n/node/192.168.5.141/vm/Ins_GetTemp/ctrl

～インスタンスの初期化～
echo start > /n/node/192.168.5.141/vm/Ins_GetTemp/ctrl

～RPCの実行～
cat /n/node/192.168.5.141/vm/Ins_GetTemp/member/Get

～引数を与える～
echo 2 > /n/node/192.168.5.141/vm/Ins_GetTemp/member/GetArg

～RPCの実行2～
cat /n/node/192.168.5.141/vm/Ins_GetTemp/member/GetArg

～インスタンスの消去、資源の解放～
echo kill > /n/node/192.168.5.141/vm/Ins_GetTemp/ctrl
```

図 3 ノードへのアクセス例

ディレクトリは，connection ディレクトリにあるので，別途後述する mount や fopen を利用し，main 関数内で初期化作業を行い利用できる状態にする必要があるが，今回は簡略化の為割愛した．4.3 章で後述する，図 4 に示す Ins_GetTemp のサンプルコードを記す．

サーバ上の C–’ プログラムにおける，クライアントや他ノードとの通信用のファイルを利用する API を示す．主に図 4 における，connection ディレクトリ内の通信用ファイルを利用するための API である．

4.2.2 クライアントによるノード資源の利用方法

クライアントからノードの資源を利用するためには，クライアントから 9P プロトコルを利用して，ノードをクライアントの持つファイルツリーの中にマウントする必要がある．つまり，ノードの提供するファイルツリーを部分木として，クライアントの持つファイルツリーの一部に張り付ける作業が必要になる．クライアントのプログラムからノードの提供するファイルを利用するときは，クライアントのファイルツリーの一部に取り込まれているため，プログラムからはローカル空間のファイルにアクセスする方法と全く同じである．ノードのファイルにアクセスがあった場合は，クライアントのファイルシステムがその旨を解釈し，通信に関しては 9P プロトコルスタックに帰着する．

マウントされたノードの資源を利用するには，4.2.1 に記した，ノードが RPC 対象手続きを仮想化して提供するファイルをインターフェースに用いる．RPC の実行を指示し結果を取得するには，クライアントのファイルツリーの一部に取り込まれている RPC 対象手続きに該当するファイルを読み込めば良い．さらに，RPC 対象の手続きが引数を必要とする場合，RPC 対象手続きに該当するファイルに引数を書き込めば良い．

ノードの資源を管理するインターフェースとして ctrl ファイルを利用する．詳しくは 4.3 章で後述する．図 4 に関して，ノードのマウント，S91 バイトコードのロード，インスタンスの初期化，RPC の実行，インスタンスの消去と資源の解放の流れを Unix のコマンドを利用した方法で記す．端末からのコマンドのみで RPC を実行できることが特徴である．これは，Unix のコマンドが 9P プロトコルに翻訳された結果であり，9P プロトコルに対応付けができれば，クライアントによらず同様の操作内容で RPC を実行することができる．例を図 3 に示す．

ノード上のファイルを読み込むためには，ノードをファイルツリーの一部にマウントする

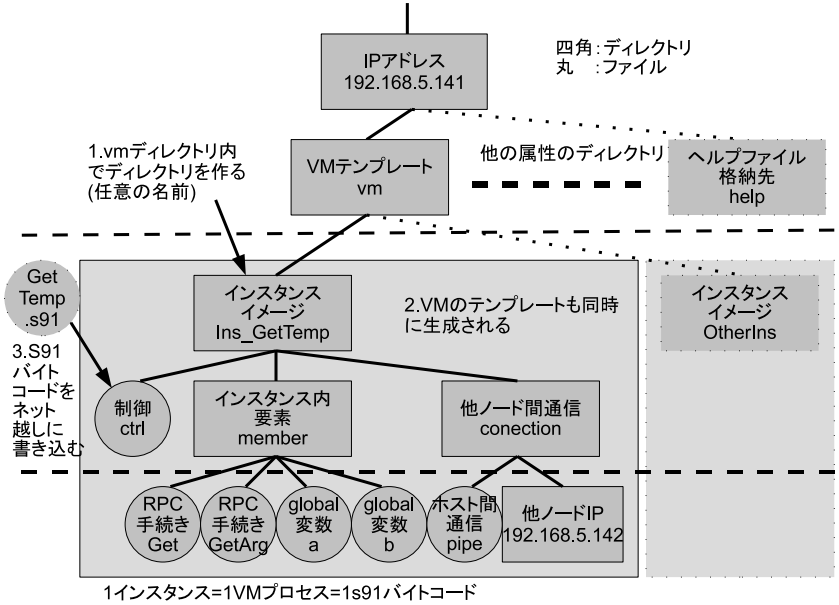

図 4 VM 提供ファイルツリー例

必要はあるが，マウントした後のファイル入出力はクライアントのファイルシステムがサポートするので，クライアントのプログラムはノードの所在を意識せずに，ローカル空間に対するファイル入出力と全く同じ操作で，RPC を実行できる．これにより，ソケットの生成指示等の煩わしい作業が不要になる．

4.3 VM を用いた資源管理方法

4.3.1 VM の提供する資源とファイル木による仮想化

クライアントのファイル空間に取り込まれる際の VM が提供する資源のイメージと役割を記す．ノードがファイル木を提供することで，クライアントのファイル木の一部となり，クライアントのプログラムからはローカル空間のファイル入出力と全く差異が無くなることが大きなメリットである．今回の設計では，1 インスタンスが 1 つの VM になるように設計を考える．VM 内で実行される単位は，一つのロードされた S91 バイトコードである．よって，1 つの S91 バイトコードが 1 つの VM によって処理される．そこで，マルチタスクを考えると，1 つの VM の実行状態を保持する物をインスタンスと考えると，1 つの S91 バイトコードが 1 つのインスタンスとして処理される．生成されるそれぞれのインスタンスは独立である．

全体の構造としては，自身の IP アドレスが記されたルートディレクトリがあり，その下に VM 上で実行する RPC のインスタンスのテンプレートである vm ディレクトリが存在する．他のディレクトリも同じ階層に同時に持つこともできるが，特殊な操作が行われるのは，vm ディレクトリに対する操作の時のみである．例として，図 4 では，help ディレクトリも IP アドレスディレクトリの下にある．

複数のインスタンス化された S91 バイトコードが，独立に vm ディレクトリの下にぶら下がることができる．実行するインスタンスを VM が自動的にコンテキストスイッチすることで，マルチタスクを実現する．

RPC を実行する際の全体の構造としては，自身の IP アドレスをルートディレクトリに，vm ディレクトリが存在し，その下に生成されたインスタンスが存在する形となる．

4.3.2 VM へのロード方法

図 4 のように，最初からツリーが構成されている訳ではない．IP アドレスが記されたディレクトリと vm ディレクトリのみが最初は存在する．この状態は，ノードに S91 バイトコードがロードされていない．

vm ディレクトリが vm にロードされる手続きのテンプレートを意味し，vm ディレクト内 (図 4 の Vm テンプレート) に新規にフォルダを作成することが，テンプレートの継承に当たる．継承された個別のテンプレート当たるフォルダには，手続きを管理するためのインターフェースが，ファイルやディレクトリの形で VM によって用意されており，それらのファイルを利用して，S91 バイトコードを書き込み，インスタンス化する．生成されたインスタンスが RPC のインターフェースとなるファイルを保持しており，このファイルに対するファイル入出力が RPC の実行となる．

例として，vm ディレクトリに VM インスタンスのディレクトリを作る．仮にその名を GetTemp とする．すると，GetTemp ディレクトリ以下にもインスタンスの管理に必要なファイルやディレクトリが生成される．

次に，GetTemp.s91 という，S91 バイトコードが記されたファイルを，先ほど生成した Ins_GetTemp ディレクトリの下にある ctrl ファイルに書き込むと，member ディレクトリ以下に RPC 手続きの対象となる関数やグローバル変数がファイルとして現れる．これは，ctrl ファイルに書き込んだ S91 バイトコードに書き込まれている情報を元に生成される．

以上により，GetTemp.s91 に記された S91 バイトコードを手続きと持つ，インスタンス

を生成できる．

4.3.3 インスタンスの提供するインターフェース

図 4 の VM インスタンスである Ins_GetTemp ディレクトリ以下に含まれるファイルやディレクトリの機能を説明する．M インスタンスである GetTemp ディレクトリ以下には，インスタンス内の要素を格納する member ディレクトリ，他ノードやホストと通信を行うためのファイルを格納した connection ディレクトリ，インスタンスを実行する VM 内での，インスタンスの状態を管理する ctrl ファイルが存在する．

member ディレクトリは，S91 バイトコード内で定義されている，RPC の対象となる関数と，グローバル変数をクライアントから参照するためのディレクトリである．member ディレクトリ直下の各ファイルに Write や Read を行うことで，RPC の対象となる関数に引数を与えたり，RPC を実行し，その戻り値を取得できる他，変数にもアクセスできる．

conection ディレクトリは，ノードを mount しているクライアントとの通信を行う為のファイルで，Write と Read を行うことでデータの授受を行う．また，ロードした S91 バイトコード内で，mount した他ノードが配置される先も connection ディレクトリである．

ctrl ファイルは，インスタンスが実行される VM の管理を行うためのインターフェースとなるファイルである．

4.3.4 VM の状態と初期化

VM 上で動作するインスタンスにも，一般的な OS 上で管理されるプロセスのように，状態を有する．状態としては，実行中を表す runnnng，実行待ち状態を表す active，休止状態を表す sleeping の三つがある．実行待ちのプログラムを，VM が自動的に切り換えて実行する．ロードされた直後のインスタンスは，休止状態に有り，各変数は初期化されていない．そこで，ロードされたインスタンスを初期化し，実行する手順を述べる．

初期化に用いるのは，ctrl ファイルである．先にも述べたが，ロードされたインスタンスは休止状態なので，ctrl ファイルに “start” 文字列を書き込み，インスタンスを実行待ち状態にする．この時に，各変数の初期化を行う．つまり，インスタンスを “start” することが初期化になる．一時的に停止しておき，状態を保持しまたまま再スタートをしたい時は “resume” を書き込むことで対応し，コードを再実行する．

4.3.5 VM の管理方法

VM の各インスタンスを管理するためのインターフェースは ctrl ファイルとなる．ctrl ファイルに特定の文字列を書き込むことで，Plan9 のように対象となるインスタンスの管理を行う．以下に，ctrl ファイルに対するコマンドをまとめる．

表 2 ctrl ファイルに対するコマンド

コマンド	動作内容
new	VM のインスタンスを生成する
override	VM に登録されている RPC の手続きを上書きする
start	VM のインスタンスを初期化し，実行可能状態にする
stop	VM のインスタンスを休止状態にする
resume	VM のインスタンスを実行可能状態にする
kill	VM のインスタンスを消去する

“new” 文字列を ctrl ファイルに書き込んだ後に，S91 バイトコードの記されたファイルを “ctrl” 文字列書き込むことでインスタンスが生成される．すでにインスタンス化され，動作している手続きを書き換えることは，インスタンスを “stop” 文字列で休止状態にした後に，“override” 文字列を ctrl ファイルに書き込み，オーバーライドしたい手続きのファイルに，その手続きの S91 バイトコードを書き込む．その後 “start” 文字列を使い，インスタンス内の変数を初期化してから実行する．インスタンス内に新しく RPC 対象の手続きを書き加えたい場合，member ディレクトリ内でファイルを作成した後に，“override” 文字列を ctrl ファイルに書き込み，そのファイルに対して S91 バイトコードを書き込む．インスタンスが不要になった場合，“kill” 文字列を ctrl ファイルに書き込み，インスタンスを削除し，資源を解放する．ctrl ファイルの読み込みは，VM インスタンスの状態を文字列として取得する．

4.4 9P プロトコルと操作の対応

サーバ上で動作する S91VM における 9P プロトコルスタックは，通常の 9P プロトコルと異なり，操作対象のファイルやディレクトリによって，RPC の実行指示や，VM インスタンスの状態を取得，などの特殊な操作を判断する必要がある．

そこで，VM インスタンスが提供するファイルやディレクトリに対する，9P プロトコルのメッセージと，特殊な操作が必要なメッセージに対する，メッセージの解釈内容を対応付ける．表 3 に 9P プロトコルのクライアントからサーバに送信される T メッセージを列挙する．内容は，T メッセージの名前と含まれるパラメータである．[] 内に記されるバイト数で各名前のデータを示す．バイト数が文字の物は可変長であることを表す．tag はメッセージを識別する為の固有の数字であり，fid はファイルディスクリプタのようなもので，fid に対してファイルに関連付けられる．ただし，複数の fid が同一ファイルを指し示すことはあるが，一つの fid が複数のファイルを差すことは無い．

表 3 から，VM インスタンスの特殊な解釈が必要なメッセージだけを表 4 にまとめる．ファイルに対する操作は，read と write と create，ディレクトリに対する操作は，create

表 3 9P プロトコル

T-mesages	パラメータ
Tversion	tag[2]msize[4]version[n]
Tauth	tag[2]afid[4]uname[s]aname[s]
Tflush	tag[2]oldtag[2]
Tattach	tag[2]fid[4]afid[4]uname[s]aname[s]
Twalk	tag[2]fid[4]newfid[4]nwname[2]nwname*(wname[s])
Topen	tag[2]fid[4]mode[1]
Tcreate	tag[2]fid[4]name[s]perm[4]mode[1]
Tread	tag[2]fid[4]offset[8]count[4]
Twrite	tag[2]fid[4]offset[8]count[4]data[count]
Tclunk	tag[2]fid[4]
Tremove	tag[2]fid[4]
Tstat	tag[2]fid[4]
Twstat	tag[2]fid[4]stat[n]

と read のみで write は出来ない．各メッセージが，対象となるファイルやディレクトリに対して実行された時の動作を示す．

- Topen

fid で指定されたファイル/ディレクトリのアクセス権の確認と I/O の準備．mode はファイルを開いた後に認める動作を示し，読み，書き，実行，に加え，ファイルの切り詰めや，ファイルを閉じるときにファイルを消すことも指示できる．ファイルの入出ロを行うときに実行されるので，Topen メッセージを受けとった時点で，S91VM が操作対象に特殊な解釈をする必要があるか判断する．

- Tcreate

fid で指定されたディレクトリに name で指定された名前のファイル/ディレクトリの作成を行う．その際に perm で指定されたパーミッションを，fid のパーミッションを考慮した上で割りつける．mode によって，ファイルの生成かディレクトリの生成かを判断する．新しくファイルが作られた場合，同時に mode で指定された内容でオープンも行われる．その際に Tcreate 内の fid の値が，新しいファイルの fid として割りつけられる．Tcreate メッセージから実行される親ディレクトリの fid が分かるので，fid を利用して Topen メッセージ同様，特殊な操作が必要か判断を行う．

- Tread

fid で指定されたファイル/ディレクトリのデータ又は所属ファイルの属性を読み取る．fid から特殊な操作が必要なファイル/ディレクトリを対象としているか判断できるため，それに応じた結果を返す．

- Twrite

fid で指定されたファイルにデータを書き込む．fid から特殊な操作が必要なファイル/ディレクトリを対象としているか判断できるため，それに応じた処理を行う．

つまり，Topen メッセージと Tcreate メッセージを受信した際に，VM が特殊な解釈をする必要があるかの判断を行う．判断結果は，ファイルやディレクトリを識別するための

表 4 9P プロトコルと動作ファイルの対応

ディレクトリ/ファイル	T-message	動作
IP アドレスディレクトリ	open	パーミッションを確認し，ファイル出力の準備を行う
	create	create はされない，最初から存在する
	read	ディレクトリの中に入っている各ファイルの stat を列挙する
vm ディレクトリ	open	パーミッションを確認し，ファイル出力の準備を行う
	create	create はされない，最初から存在する
	read	ディレクトリの中に入っている各ファイルの stat を列挙する
インスタンスディレクトリ	open	パーミッションを確認し，ファイル出力の準備を行う
	create	テンプレートの継承
	read	ディレクトリの中に入っている各ファイルの stat を列挙する
ctrl ファイル	open	パーミッションを確認し，ファイル入出力の準備を行う
	create	create はされない，自動的に生成される
	write	インスタンスの状態を設定する/インスタンス化
	read	インスタンスの状態を出力する
member ディレクトリ	open	パーミッションを確認し，ファイル出力の準備を行う
	create	create はされない，自動的に生成される
	read	ディレクトリの中に入っている各ファイルの stat を列挙する
RPC 手続きファイル	open	パーミッションを確認し，ファイル入出力の準備を行う
	create	手続きオブジェクトの追加生成
	write	引数の受け渡し/バイトコードの書き込みでインスタンス化
	read	RPC の実行と戻り値の取得
グローバル変数ファイル	open	パーミッションを確認し，ファイル入出力の準備を行う
	create	グローバル変数の追加
	write	グローバル変数の書き換え
	read	グローバル変数の読み出し
connection ディレクトリ	open	パーミッションを確認し，ファイル出力の準備を行う
	create	create はされない，自動的に生成される
	read	ディレクトリの中に入っている各ファイルの stat を列挙する
pipe ファイル	open	パーミッションを確認し，ファイル入出力の準備を行う
	create	名前付きパイプの生成
	write	データを送る
	read	データの取得する

fid と対応付けることができる．Tread メッセージや Twrite メッセージは，fid に割りつけられている特殊な解釈が必要かを判断し，求められる操作を行う．

表 4 に示されたもの以外に，connection ディレクトリでのディレクトリの生成と，member ディレクトリでのディレクトリの作成は，それぞれの親ディレクトリの特徴を継承するものとする．つまり，member ディレクトリ内に submember ディレクトリを作成したとき，submember ディレクトリ内でファイルを生成しても，member ディレクトリ内でファイルを生成した時と同様に，インスタンス内に新たなメソッドが追加されたとみなす．それ以外は，本処理系で特殊な解釈は行わず，通常の 9P プロトコルの持つ意味として，ファイルないしはディレクトリの生成を行う．

5. 実　　装

今回は ColdFire MCF52233 をターゲットに実装を行った．同じマイコン上に実装されている SilentC[8] インタプリタとネイティブコードと動作性能を比較する．

また，メモリ使用量に関して，S91 の必要とする ROM サイズは 25KB 程度であり，使用 RAM サイズは S91 が 15KB 程度となっており，組込みの 32bitCPU には十分である．

3 重ループは各要素 10 の中でインクリメントを行い，再帰呼び出しは 10 までのフィボナッチ数を求め，和演算は 1〜10 の和を計算した．全体として，本研究の方式はネイティブコードに比べ 1000 倍近い実行時間がかかるが，インタプリタである SilentC の 10 分の 1 程度の実行時間で処理できる．

また，サーバ上で動作する S91VM の 9P プロトコルスタックを一部を実装し，動作確認を行った．具体的には，サーバでは予め VM に登録されているファイル情報を用い，クライアントにあたる Linux の mount コマンドや，ls コマンド，cd コマンドに対する応答を実現した．

6. おわりに

本研究では分散システムの通信プロトコルに 9P プロトコルを用い，ノードの手続きをローカル空間でのファイル入出力のように扱う，位置透過性の高い分散システムの構築法を提案した．ノード上の手続きをロードすることも，RPC の実行もファイルに対する入出力を通して行える．VM 上で動作するインスタンスを管理する為のインターフェースにもファイルを用い，Plan9 のような文字列により，VM の実行を制御できる．ソケットのような低水準な入出力を用いることなく，クライアントからもノードからもファイル入出力のインターフェースを用いて通信できる．また，提案した分散システムを実現するために，S91VM と C–’ 言語の処理系の試作を行い，ノードで動作する処理系として，十分な実行速度を実現した．

主な課題は，9P プロトコルの実装を VM 上で進め，すべての 9P プロトコルに対応し，プロトコルに応じた処理を実装することである．また，C–’ コンパイラから生成した S91 バイトコードをネットワークにより転送し，VM 上で利用できるようにすることである．サーバとしての機能を拡充したのちには，クライアントとしての機能を持たせることで，9P プロトコルをサポートする処理系との連携を強化することである．

表 5　MCF52233 上での実行時間

	本処理系	SilentC	ネイティブ
3 重ループ	0.12[s]	1.19[s]	0.13[ms]
再帰呼び出し	0.03[s]	0.29[s]	0.08[ms]
和演算	0.18[ms]	1.25[ms]	0.2[μs]

参 考 文 献

1) Bell-labs: Plan 9 from Bell Labs Fourth Edition , plan9.bell-labs.com/plan9/
2) Bell-labs: Plan 9 File Protocol, 9P , plan9.bell-labs.com/sys/man/5/INDEX.html, 2008.
3) Common Object Request Broker Architecture: Core Specification: Object Management Group, http://www.omg.org/cgi-bin/doc?formal/04-03-01.pdf
4) Q. Cao, T. Abdelzaher, J. Stankovic, K. Whitehouse, L. Luo : Declarative Tracepoints: A Programmable and Application Independent Debugging System for Wireless Sensor Networks, In Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (ACM/IEEE SenSys), 2008.
5) Crossbow: Wireless Module Portfolio, www.xbow.com/Products/productdetails.aspx?sid=156 .
6) vita nuova: Inferno, www.vitanuova.com/inferno/
7) 佐藤好秀 丸山勝巳: 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 (システムソフトウェア構成法), IPSJ SIG Notes 2007(10) pp.9-16, 20070130.
8) SilentSystem: OS1 プログラミングマニュアル, www.silentsystem.jp/download/OS-1_Programing_06.pdf (2007).
9) Brian Wilson Kernighan, Dennis MacAlistair Ritchie, 石田 晴久: プログラミング言語 C 第 2 版 ANSI 規格準拠, 共立出版, 1989.
10) Philip Levis, David Culler: Mate: a tiny virtual machine for sensor networks ACM SIGOPS Operating Systems Review Volume 36 pp.85-95, 2002.