

動的なメディア選択が可能な Mobile IP の設計と実装

小林 勝, 中島 達夫

北陸先端科学技術大学院大学 情報科学研究科

概要

計算機の小型化や無線技術の発展によりいつでもどこからでもネットワークに接続することが可能になってきた。このような移動計算機環境では移動や通信メディアの変化による IP アドレスの動的な変化というこれまでの計算機環境とは異なる問題が発生している。この問題に対処するため、IETF Mobile IP のようなホストの移動を可能とするプロトコルが開発されているが、完全には解決できていない。本稿ではシステムソフトウェアからの対処法として動的に通信メディアを選択する機構を構築し、IETF Mobile IP と組み合わせることで移動計算機環境に適応したネットワークを実現する。

Design and Implementation of Mobile IP with Dynamic Network Media Selection

Masaru Kobayashi, Tatuso Nakajima
Japan Advanced Institute of Science and Technology

Abstract

Small computers such as PDAs and portable computers and wireless networks have made possible to connect these computers to network at any time from everywhere. In mobile computing environments, dynamic changes of the IP address by moving computers or changing of network media. This causes a new problem that we do not have before. A new network protocol such as Mobile IP to take into account this problem has been developing, but the problem cannot be completely solved. In this paper, we develop a system with dynamic network selection for solving the problem. This system adopts with IETF Mobile IP and enables network system adaptive to mobile computing environments.

1 はじめに

小型化により計算機を携帯することが可能になり、研究室などの屋内だけでなく野外を含む移動先でも計算機を利用する事が可能になった。また無線を中心とするネットワーク技術の進歩によりネットワークへ接続可能な領域が大幅に広がった。この携帯可能な計算機とネットワーク技術を用いる事によりいつでもどこでもネットワーク上のサービスや他の計算機を含む計算機資源を利用する事が可能になっている。

例えば、屋内の研究室などではイーサネットや FDDI、ATM などの高速な有線のネットワークを使用して、会議などで別の部屋へ移動する時には無線 LAN を使用する。さらに屋外では PHS や携帯電話を使ってネットワークに接続する、などの使い方が考えられる。このように近年の移動計算機環境は各種の通信メディアを駆使する事によりいつでもどこでもネットワークに接続する事ができる。

しかし、通信メディアを使い分けて移動しながらネットワークに接続する事は、使用する通信メディアやネットワークとの接続位置が移動先やその時の環境によって常に変化する事を意味する。これは通信メディアやネットワークの接続位置が変化しないというこれまでの計算機環境とは大きく異なる。そのため、従来のネットワークプロトコルやオペレーティングシステムのネットワークサポートでは移動計算機環境に対応し切れない部分が生じている。

このような問題に対応するため、スタンフォード大学の

MosquitoNet[1] などでは移動計算機環境のためのネットワークプロトコルやシステムソフトウェアの研究が進められている。現在、ネットワークプロトコルの対処としてネットワーク上での移動を可能とするプロトコルが幾つか提案されている。しかし、それだけでは解決できない問題が残されている。

そこで、我々はシステムソフトウェア側の対応として動的に通信メディアを選択する機構を RT-Mach(Real-Time Mach[5])上に構築した。このシステムは移動を可能とするプロトコルの一つである IETF Mobile IP[2] や、接続されている通信メディアなどの計算機資源の情報や位置情報を管理する環境サーバ[8]とともに動作し、移動計算機環境のネットワークの問題点を解決する。本稿ではこの通信メディア選択機構の設計、実装について検討する。¹

2 移動計算機環境の特徴と問題点

2.1 多様な通信メディア

移動計算機環境で利用可能な通信メディアは多様である。イーサネットや FDDI のような有線のメディアは高速だが屋内の特定の場所でしか利用できない。無線 LAN は有線のメディアに比べれば低速であるが、移動が可能である。しかし接続可能な範囲は基地局から数 100m の距離内に限られる。PHS を

¹IETF Mobile IP の処理系はカーネギーメロン大学の Monarch プロジェクトで実装されたものをベースにしている。

利用する PIAFS は都市部で、携帯電話はより広範囲で利用可能だが、速度は大幅に低下する。このようにいつでも利用可能な通信メディアは存在するが、一つのメディアですべての要求に対処できるわけではない。したがってネットワークに接続したい場所によって通信メディアを選択する必要がある。

現在の移動計算機は動的に抜き差しが可能な PC カードを利用する事ができる。移動計算機向けの通信メディアはイーサネットなどの有線のメディアでも無線 LAN や PIAFS などの無線のメディアでもこの PC カードにより実現されている。これにより計算機の動作中でも必要に応じて通信メディアの選択が可能になっている。たとえば、研究室でイーサネットのような有線のメディアを使用している時に会議などで別の部屋に移動しなければならなくなった場合は、イーサネットの PC カードを抜いて無線 LAN の PC カードを挿入すれば通信メディアをイーサネットから無線 LAN に切り替える事ができる。さらに無線 LAN の利用可能な範囲から出てしまった場合は、無線 LAN の PC カードを PIAFS の PC カードや携帯電話用の PC カードに取り替えればネットワークへの接続を維持する事ができる。

2.2 移動計算機環境の問題点

このように移動計算機環境では PC カードと各種通信メディアを組み合わせる事でハードウェア的にはいつでもネットワークに接続する事が可能になっている。しかし、同時に計算機の動作中に IP アドレスが動的に変化するというこれまでにない問題を生じさせている。これには二つの要因がある。

(1) 計算機の移動

TCP/IP の IP アドレスはネットワークを識別するネットワーク識別子と計算機を識別するホスト識別子の組みからなる。計算機が移動して接続するネットワークが変わると移動前の IP アドレスは使用できなくなり、接続先のネットワークで新しい IP アドレスを割り当てなければならない。しかし、ネットワーク上のサービスやアプリケーションが通信に使用するので、計算機が移動して IP アドレスが変化するとアプリケーションは通信を続けることができない。

このように現在用いられているネットワークプロトコルの TCP/IP は計算機が移動する事を想定していない。これまでの計算機は大きくて重かったので、簡単に移動させる事ができなかった。そのため移動に伴う IP アドレスの変化は問題にならなかった。

(2) 通信メディアの切替

TCP/IP ではネットワークアドレスは計算機ではなく通信メディアに対して与えられる。このため通信メディアの変更は IP アドレスの変更を意味する。通信メディア変更後も同じネットワークに接続する場合でも計算機側ではどのネットワークに接続するかは確定できないので、ユーザが IP アドレスを指定するか DHCP (Dynamic Host Configuration Protocol) [4] で外部から割り当てる必要がある。

このように IP アドレスは計算機を識別するための識別子というよりはネットワークから見た計算機との接続点となっている。したがって通信メディアの変更はネットワークへの接続点の変更であり、計算機の移動と同じ問題が発生する。

2.3 IETF Mobile IP

計算機の移動に伴う問題点に対処するためのプロトコルとして IETF Mobile IP や VIP [3] がある。ここでは IETF Mobile IP の概要について述べる。

IETF Mobile IP では移動する計算機は移動ホストと呼ばれ、ホームネットワークとホームネットワーク上の IP アドレスであるホームアドレスを持つ。ホームネットワークにはホームエージェントがあり、移動ホストのホームアドレスを管理して

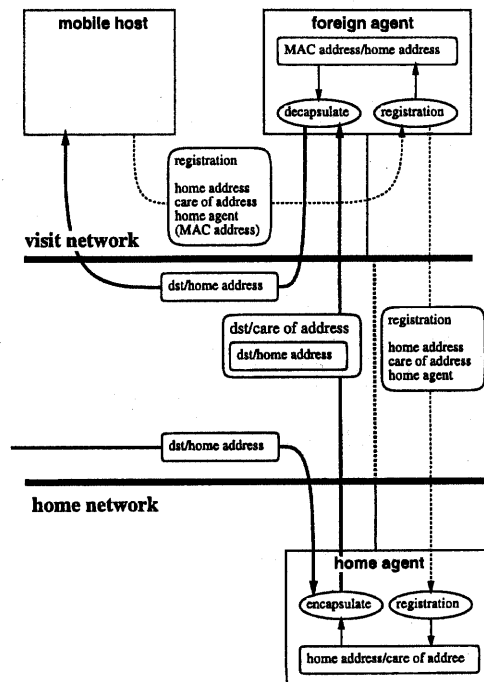


図 1: IETF Mobile IP の動作

いる。移動ホストはホームネットワークから他のネットワークへ移動する事ができる。移動ホストが移動したネットワークにはフォワーリングエージェントが配置されている。(図 1)

IETF Mobile IP の基本的な動作は他のネットワークに移動した移動ホスト宛の IP パケットをホームエージェントが移動ホストへ転送することである。移動ホストはホームネットワークから他のネットワークに移動すると、移動先のネットワーク上で気付アドレス (care of address) を取得する。気付アドレスの取得方法は移動先のネットワーク上のフォワーリングエージェントから割り当ててもらふ場合と、移動ホスト自身が DHCP などを用いて移動先のネットワーク上の IP アドレスを確保しそれを気付アドレスとする場合がある。気付アドレスを確保した移動ホストは気付アドレスをホームエージェントに登録する。

ホームエージェントはホームネットワークに送られてきたホームアドレス宛の IP パケットを移動ホストの代わりに受け取り、IP in IP のカプセル化をおこなって移動ホストに登録した気付アドレスに転送する。気付アドレスがフォワーリングエージェントの場合はフォワーリングエージェントがカプセル化された移動ホスト宛の IP パケットを取り出し、移動ホストに転送する。気付アドレスが移動ホストが確保したアドレスの場合は移動ホストが自分でカプセル化された IP パケットを取り出す。このような手順でホームアドレス宛の IP パケットを転送する事により移動ホストは移動先でも自分宛の IP パケットを受け取る事ができる。移動ホストが送信する IP パケットのソースアドレスは移動先でもホームアドレスを用いる。これにより移動ホストの通信相手は常にホームアドレス宛に IP パケットを送るので、移動ホストはどのネットワークに移動しても通信を続ける事ができる。

このように IETF Mobile IP ではホームアドレスはネットワークインターフェースに割り付けられたアドレスではなく

移動ホストの識別子として働く。ネットワークへの接続点としてのIPアドレスは気付アドレスとして移動したネットワーク上でその都度確保される。

2.4 IETF Mobile IP の問題点

IENTF Mobile IP ではホームエージェントが移動ホストへIP パケットを転送する事で計算機の移動に伴うネットワークアドレスの変化に対応している。しかし、その仕様には以下に示すいくつかの問題が残されている。

(1) フォーリンエージェントから移動ホストへの転送

移動ホストの移動先のネットワークで、フォーリンエージェントから移動ホストへ転送されるIP パケットの宛て先アドレスはホームアドレスとなっている。この場合ホームアドレスは別のネットワーク上のアドレスなので、通常のIP パケットのルーティングを用いる事ができない。このためフォーリンエージェントはリンク層を直接制御して移動ホストにパケットを転送しなければならない。リンク層がイーサネットの場合、フォーリンエージェントは移動ホストのMACアドレスを指定してパケットを転送する。

移動ホストがPPP で接続する場合、移動ホストはPPP サーバを介してフォーリンエージェントの配置されたネットワークに接続される。そのためフォーリンエージェントがホームアドレス宛のパケットを転送する先はPPP サーバになる。したがってPPP サーバもIENTF Mobile IP を認識しなければならない。

このようにフォーリンエージェントから移動ホストへのIP パケットの転送はリンク層の制御が必要になり、移動ホストが使用する通信メディアごとにIENTF Mobile IP に対応する必要がある。

(2) ホームアドレスと気付アドレス

気付アドレスを移動ホスト自身が確保する場合、ホームエージェントから転送されてくるカプセル化IP パケットを受け取るために、移動ホストの通信メディアは気付アドレスで動作しなければならない。一方移動計算機から通信相手のホストへ送信されるIP パケットのソースアドレスには気付アドレスではなくホームアドレスを使用しなければならない。そのため、ネットワークに接続された通信メディアはネットワークからのARP などに対しては気付アドレスを持ったホストとして動作し、送信されるIP パケットにはホームアドレスを持ったホストとして動作するという変則的な動作が必要になる。

2.5 問題点の検討

移動計算機のネットワークでは計算機の識別子とネットワークへの接続点としての識別子の分離が必要になる。ネットワークプロトコルはIENTF Mobile IP などのホストの移動に対応した拡張が行われ、計算機の識別子とネットワークへの接続点としての識別子の分離が行われている。

一方、現在のオペレーティングシステムのネットワークコードは計算機は移動せず、通信メディアなどのハードウェア構成が動的に変化する事は無いという前提で設計されている。ネットワークへの接続点である通信メディアは動作中に取り替えられる事はないので、IP アドレスは通信メディアに直接割り当てられる。IP アドレスはネットワークへの接続点であり、ホストの識別子とはなっていない。IENTF Mobile IP の残された問題点もネットワークコードがホストの識別子とネットワークへの接続点の識別子を分離していない事に起因すると考えられる。

3. 通信メディア選択機構の設計と実装

3.1 プロトコルスタックと通信メディアの分離

現在のネットワークコードでホストの識別子とネットワークへの接続点としての識別子の分離することは、一つの通信

メディアに複数のIP アドレスを割り当てる事になる。これはプロキシARP によってネットワークに対しては通信メディアに割り当てられたIP アドレスのほかに別のIP アドレスとしても見せたり、トンネルデバイスを用いて出力されるIP パケットのソースアドレスを書き換えるなどの方法で対処する事ができる。これらの方法は現在の枠内での解決方法ではあるが、ネットワークコードが想定している使い方ではない。

問題点はプロトコルと通信メディアが密に結び付けられているためIP アドレスが通信メディアに直接割り当てられてしまい、ホストの識別子を持つレベルが無い事である。そこでプロトコルと通信メディアを明確に分離し、プロトコルの下位に通信メディアを管理する層を配置する。プロトコル側はホストの識別子としてのIP アドレス、通信メディア側にはネットワークへの接続点としてのIP アドレスを与える。具体的にはプロトコルにIENTF Mobile IP を用いる事を前提として、プロトコル側のIP アドレスはIENTF Mobile IP のホームアドレスを与え、通信メディア側のIP アドレスは接続先でDHCP などでの都度割り当てる。これによりホストの識別子とネットワーク接続点の識別子を分離する。

通信メディアを管理する層は複数の通信メディアを管理し、状況やユーザの要求によって最適な通信メディアを選択してネットワークに接続する。管理する通信メディアにはイーサネットやPPP などがある。ネットワークへの接続に必要なIP アドレスの確保やARP などの処理はこの層で行なう。各通信メディアの違いはこの通信メディア選択機構で吸収し、プロトコルからは単一のネットワークインターフェースとして仮想化する。プロトコルはこの通信メディア選択機構にホスト識別子としてホームアドレスを与えたネットワークインターフェースとして使用する。

通信メディア選択機構はプロトコルに対してフォーリンエージェントとしても動作する。気付アドレスに通信メディア選択機構が確保したIP アドレスを用いる事でホームエージェントから移動ホストへのIP パケットの転送をIP を用いて行なう事ができる。ホームエージェントから転送されたカプセル化されたパケットは、通信メディア選択機構がカプセルから取り出して上位のプロトコルに渡す。これにより、プロトコルは変則的な動作を行なう必要がなくなる。

3.2 実装上の検討

通信メディア選択機構はプロトコルスタックとネットワークデバイスドライバの間に位置する。実装する場所としてはプロトコルスタックやデバイスドライバに組み込む方法とサーバとして実装する方法がある。UNIX の場合、プロトコルスタックはデバイスドライバとともにカーネル内に実装されている。前者の方法の場合は通信メディア選択機構をカーネル内に実装することになる。カーネル内のプロトコルスタックとデバイスドライバは比較的密に結び付いており、この部分に通信メディア選択機構を組み込むには大きな変更が必要になる。

一方、今回実装の対象とするマイクロカーネル方式のRT-Mach の場合、プロトコルスタックはユーザレベルのサーバであるLites[6] 内に、デバイスドライバはマイクロカーネル内に実装されている。したがって通信メディア選択機構はユーザレベルのサーバであるLites(図2) とマイクロカーネルのどちらかに組み込むことになる。プロトコルスタックとデバイスドライバがサーバとカーネルに分離しているのでUNIX ほど密な関係ではないが、その間のインターフェースの変更は必要になるのでLites やカーネルの大きな変更が必要と考えられる。

サーバ方式の場合はユーザレベルのプロセスとして実装する。プロトコルスタックに対してはデバイス側がユーザプロセスから読み書き可能なネットワークインターフェースとして接続し、デバイスドライバに対してはカーネルに直接接続する。これにより既存のプロトコルスタックなどを変更せず

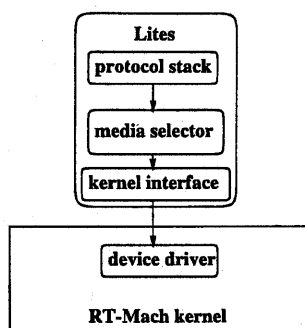


図 2: 通信メディア選択機構の Lites への実装

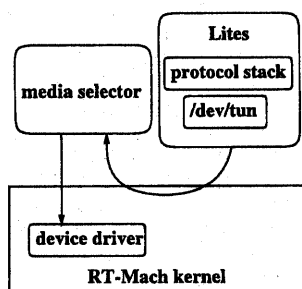


図 3: サーバ方式での通信メディア選択機構

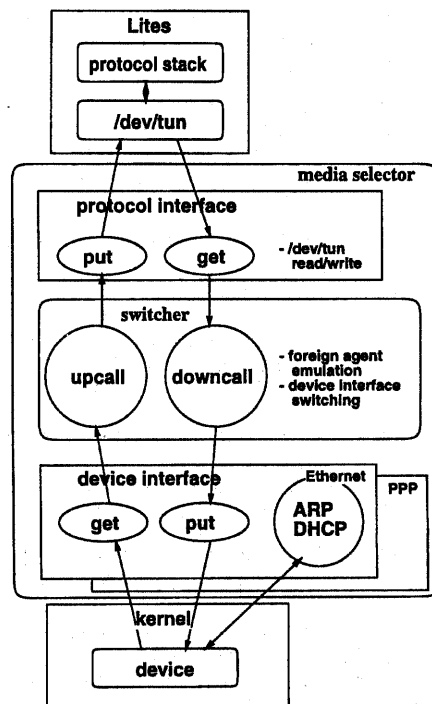


図 4: システムの構成

に機能を組み込む事が出来る。また、サーバ方式は既存の部分から完全に独立しているので変更が容易である。デバイス側を読み書き可能なネットワークインターフェースは UNIX、Lites とともにトンネルデバイスとして実装されている。デバイスドライバとのインターフェースにはバケットフィルタと呼ばれるインターフェースがカーネル（デバイスドライバ）内に実装されている。

そこで、今回の実装では既存の部分を変更する必要があるサーバ方式（図 3）で実装を行なう事にした。

3.3 システムの構成

通信メディア選択機構は図 4 のようにプロトコルインターフェース、デバイスインターフェース、スイッチ部から構成される。プロトコルインターフェースはプロトコルスタックとの入出力を行ない、デバイスインターフェースは通信メディアのデバイスドライバとの入出力を行なう。スイッチ部はデバイスインターフェースとプロトコルインターフェースを結び付ける。デバイスインターフェースはデバイスドライバの種類（通信メディアの種類）ごとに定義され、スイッチ部がプロトコルインターフェースに接続するデバイスインターフェースを切り替える事で複数の通信メディアを使い分ける事が出来る。プロトコルインターフェース、デバイスインターフェースとスイッチ部間でインターフェースが定義されており、スイッチ部はどのプロトコルインターフェース、デバイスインターフェースとも同じ方法で接続する事が出来る。

プロトコルインターフェースの主な機能はプロトコルスタックとの入出力である。現在はトンネルデバイスを想定しており、トンネルデバイスとの入出力のほかに初期化や終了処理を想定したインターフェースを定義している。各機能はスイッチ部から呼び出される関数として実現されている。

デバイスインターフェースはデバイスドライバとの入出力のほかにネットワークとの接続も管理する。デバイスドライバとの入出力には RT-Mach マイクロカーネルのバケットフィルタを使ってデバイスドライバと直接接続して行なう。入出力やバケットフィルタの設定などの機能はスイッチ部から呼ばれる関数として実現されている。一方、ネットワークの接続の管理はスレッドとして実現されている。スレッドは ARP、DHCP の機能のほかにネットワークとの接続状態の監視機能も持っている。ネットワークに接続されてから IP アドレスを確保するまでの処理はこのスレッドが行なっている。デバイスインターフェースとスイッチ部の間では IP パケットをやり取りしており、リンク層のための処理はすべてデバイスインターフェース側で行なっている。このためスイッチ部はリンク層を意識する必要はなく、イーサネットと PPP の同等に扱う事が出来る。

スイッチ部はプロトコルインターフェースとデバイスインターフェースを結び付け IP パケットの入出力を行なう。スイッチ部はアップコール用とダウンコール用の二つのスレッドを持っており、IP パケットの入出力を平行して行なっている。アップコールスレッドはデバイスインターフェースを用いて IP パケットを取り込みプロトコルインターフェースを用いてプロトコルスタックに渡す。ダウンコールスレッドは反対の動作を行なう。スイッチ部はそのほかに IETF Mobile IP のフォーリンエージェントとしての機能を持っている。移動ホストがネットワークへの接続の監視やフォーリンエージェントの探索に用いる拡張された ICMP ルータディスカバリへの応答、ホームエージェントとの間のレジストレーションメッセージの中継を行なう。スイッチ部はデバイスインターフェースを切り替えるためにデバイスインターフェースの状態を監視している。切替の時にはアップコールとダウンコールのスレッド

で同時に切り替えなければならないので、ミューテックスとコンディション変数を用いた同期機構が用意されている。

4 設計、実装に関する考察

本研究では通信メディア選択機構をサーバ方式で RT-Mach 上に設計、実装した。本節では今回の設計、実装を他の方式や UNIX を対象とした場合と比較する事で問題点や利点について考察する。

4.1 システムの構成

本研究で設計、実装した通信メディア選択機構はネットワークプロトコルスタックとネットワークデバイスドライバの間に挿入される。実現方法としては既存のモジュールに追加することで直接挿入する方法と、本体はサーバとして作成し既存のモジュールにはプロトコルスタックとデバイスドライバの間に本体への迂回路のみを組み込む方法が考えられる。

(1) 既存のモジュールへ追加する方法

プロトコルからデバイスドライバまでは処理を高速化するために密接に作られている。実際、ネットワークデバイスドライバのカーネルインターフェースは RT-Mach でも UNIX でもネットワーク専用になっている。とくに UNIX ではプロトコルスタックもカーネル内に実装されていてデバイスドライバとプロトコルスタックは直接結び付いている。通信メディア選択機構を直接この部分に挿入する事は、カーネルの大幅な変更が必要になると考えられる。

マイクロカーネルである RT-Mach の場合はプロトコルとデバイスドライバはユーザレベルのサーバである Lites とマイクロカーネルに分けられている。通信メディア選択機構をプロトコルとデバイスドライバの間に挿入する事を考えた場合、Lites のプロトコルスタックの下位に組み込む、サーバとして Lites とマイクロカーネルの間に挿入する、マイクロカーネル内のデバイスドライバの上位に組み込む、の3種類が考えられる。

Lites またはカーネルに配置する場合、Lites とカーネル間のインターフェースとプロトコルスタック、またはデバイスドライバの間に挿入することになる(図2)。この時 Lites とカーネルの間のインターフェースを変更しないようにすると、UNIX と同じ様に密接に作られた部分に追加する場合と変わらない可能性がある。Lites はユーザレベルのサーバなので UNIX のようにカーネルを変更する場合よりは軽い作業のように思われるが、Lites は 4.4BSD のサービスを提供するかなり大きなサーバなので実際の作業は UNIX の場合とあまり変わらない可能性が高い。Lites とカーネルの間のインターフェースを変更する場合は Lites またはカーネルの変更は小さくなるかもしれないが、Lites とカーネルの両方を変更しなければならない事も考えられる。したがって、既存のモジュールに追加する方法は RT-Mach でもかなり重い作業になると予想される。

(2) サーバによる方法

サーバ方式では、プロトコルスタックとデバイスドライバの間でパケットを横取りし必要な処理を行なった後プロトコルスタックまたはデバイスドライバに出力する。プロトコルスタックにはデバイス側をユーザプロセスから読み書き可能なトンネルデバイスという仮想ネットワークインターフェースが実装されている。一方デバイスドライバ側はパケットフィルタと呼ばれるデバイスドライバと直接接続できるインターフェースが既に存在する。これらのインターフェースは UNIX、RT-Mach とともに持っている。UNIX の場合はこれらのインターフェースを使って通信メディア選択機構のサーバを実装する事が出来る(図5)。

この方法ではプロトコルスタックやデバイスドライバに必要なインターフェースが既に実装されているので、既存のモジュールは変更する必要がない。さらにサーバはユーザプロ

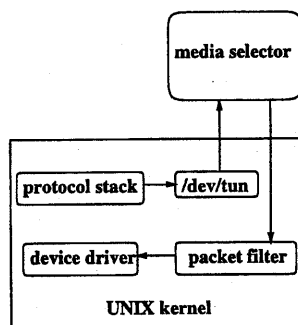


図5: UNIXでのサーバ方式による通信メディア選択機構

セスとして実現されるのでカーネル内に組み込む場合に比べてはるかに軽い作業ですむと予想される。

RT-Mach の場合はすでに Lites とマイクロカーネルという形でプロトコルスタックとデバイスドライバが分離されているので、UNIX のようなトンネルデバイスとパケットフィルタを用いる方法のほかに Lites とマイクロカーネルの間に配置する方法も考えられる。しかし、Lites とマイクロカーネルの間は Lites が個々のデバイスドライバと接続されるので、ここにネットワークデバイスを統一的に扱おうとする通信メディア選択機構を配置する事は非常に難しい。少なくとも Lites 側のインターフェースも変更が必要になる。

一方トンネルデバイスとパケットフィルタを用いる方法は UNIX と同じ様に既存のモジュールの変更は必要ない。既存のインターフェースを用いてサーバを実装するのならば UNIX でもマイクロカーネルでも一つのユーザプロセスなので、どちらの OS でも大きな違いはないと思われる。

以上の検討から、密接に結び付いたモジュール間に挿入する通信メディア選択機構は、直接組み込むよりはサーバ方式の方が変更量や作業は軽くなると考えられる。また、RT-Mach の場合、Lites とカーネルの間に配置する方法もあるが、コストは小さくない。マイクロカーネルは各機能がサーバとして実現され機能の変更や拡張を行ないやすいといわれていたが、サーバ間にまたがったりインターフェースをそのまま使えないような場合のコストは決して小さくないと考えられる。

サーバ方式の問題点としてユーザモード、カーネルモードの切替が増えオーバーヘッドが増加する事がある。UNIX の場合、カーネル内のプロトコルの処理の間に一度ユーザ空間に出て再びカーネル空間に戻ってくることになり、空間の切替が2回発生する。RT-Mach の場合も、Lites からサーバへの切替に2回、サーバからマイクロカーネルに1回の切替が必要であり、従来の Lites からマイクロカーネルへの1回から3回になっている。どちらもオーバーヘッドは増加するが、増加の度合は RT-Mach の方が小さい。

4.2 実装

通信メディア選択機構はプロトコルスタックからの IP パケット、ネットワークデバイスドライバからのフレーム、ネットワークとの接続のための ARP への応答などの処理を同時に行なわなければならない。このような複数の入出力を同時に扱う方法は、UNIX で提供されている select システムコールのような入出力が可能になったかどうかを同時にチェックできる機能を使う方法と、RT-Mach で提供されているスレッドを使う方法がある。

select システムコールを用いる場合は入出力をイベントと

見立てて、select システムコールでイベントを検出してイベント毎の処理を行なうイベント駆動型の設計を行なうことになる。この時、一つのデータの流の中の入出力を別々のイベントとして扱わなければならない場合がある。この場合一つのデータの流にもかわらず入出力は別々のイベントに対する処理として設計しなければならない。さらに、複数のイベントを同時に扱う必要上、イベント処理の中で待ち合わせを行なうことができず、待ち合わせ自体もイベントや内部状態として持たなければならない。このため、イベントや内部状態とそれに対する処理はかなり細かく定義しなければならない。

スレッドを用いる場合は入出力データの流毎にスレッドを割り当てて並列に処理させる。これにより個々のデータの流に対してシーケンシャルに処理する事が出来るので、各データの流が独立している場合はイベント駆動型に比べ設計は容易である。しかし、データの流が密接に関係している場合は複数のスレッド間で同期が必要になる。処理の流れの中で同期が必要な場所が多くなると各処理毎の同期の関係が複雑になり、デッドロックの可能性が生じる。これを避けるためには処理の流れを細かく分解し、注意深く同期をとらなければならない。このため、複雑な同期を行わなければならないような密接な関係を持つデータの流が存在する場合はスレッドの利点がなくなってしまう。

select システムコールとスレッドの比較を元に、通信メディア選択機構の実装方法を検討する。検討項目には実装の容易性と拡張性があげられる。

(1) 実装の容易性

両方式を比較すると、複雑に関係した複数の流がある場合はイベント駆動型が有利で、関係があまりない場合はスレッドの方が有利と考えられる。通信メディア選択機構の場合、データの流はプロトコルスタックとデバイスドライバの流れ、各デバイス毎の ARP や DHCP などのデバイス毎の処理がある。これらの流はほぼ独立しており、別々に扱っても支障が無い。プロトコルスタックとデバイスドライバの流れにはアップコールとダウンコールの二つの流れがある。デバイス切替の時は同時に切り替えなければならないので同期が必要になるが、必要な同期はこれだけなのでたいした問題にはならない。したがって、通信メディア選択機構の場合はスレッドを用いた設計の方が select システムコールを用いたイベント駆動型の設計より有利と考えられる。

(2) 拡張性

通信メディア選択機構は複数の特性の異なる通信メディアを扱わなければならない。例えばイーサネットと PIAPS などを使った PPP ではデバイスドライバとのインターフェースがまったく異なり、また入出力の手順も異なる。また、イーサネットの場合は ARP と DHCP の処理が必用であるが、PPP の場合 ARP は必要なく IP アドレスの割り当ては回線接続の過程で行われる。このようにイーサネットと PPP では処理手順がまったく異なり、イベントとしてみた時にはまったく別々に扱わなければならない。したがって、イーサネットのみで設計された通信メディア選択機構に PPP を追加する場合、イベント駆動の設計ではイベントを検出する中心部分を変更しなければならないだけでなく、イベント全体の再設計が必要になる可能性もある。しかし、データの流れとしては独立しているのでスレッドが使える場合は別々のスレッドに割り当てれば良く、新しいデバイスを追加する場合もデバイスインターフェースの追加だけすむと予想される。

4.3 結論

以上、RT-Mach 上の通信メディア選択機構の設計と実装について UNIX との比較も含めて検討を行った。RT-Mach のようなマイクロカーネルは機能をサーバとして持つので拡張性が高いといわれているが、通信メディア選択機構のような

サーバやカーネルとのインターフェースの変更が必要な場合やカーネルやサーバの変更が必要な場合は拡張に必要な労力は UNIX のようなモノリシックな OS とあまり変わらないと考えられる。本研究で採用したサーバ方式では既存のインターフェースを利用した事もあり、RT-Mach と UNIX では余り差がなさそうである。サーバ方式はパフォーマンス上は不利であるが、管理するデバイスの追加など今後の拡張を考えるとサーバ方式の方が好ましい。

一方複数の入出力を扱うことを考えると、通信メディア選択機構が扱うデータの流は独立性が高い事もありスレッドを前提とした設計の方がはるかに有利と考えられる。最近の UNIX はユーザレベルスレッドを持っているが、デバイスドライバのようなカーネル内での待ちが発生する場合はユーザレベルスレッドでは全スレッドが停止してしまうので、通信メディア選択機構には不十分である。この点ではカーネルレベルスレッドの使える RT-Mach の方が有利と思われる。また、通信メディア選択機構が扱うデバイスの追加や機能の追加を考える時、スレッドは各モジュールの独立性を高めるのに役立っている。

全体として、システム構成上は RT-Mach と UNIX では余り差が無いが、実装上は RT-Mach が有利と思われる。通信メディア選択機構は今後対応する通信メディアの拡大などの拡張が予想される。この事も含めると通信メディア選択機構の設計と実装は妥当な物と思われる。

5 まとめ

本研究では移動計算機環境のネットワークの問題点を解決するために IETF Mobile IP と組み合わせて使用する通信メディア選択機構の設計と実装を行った。UNIX を比較対象として設計や実装の検討を行ない、RT-Mach の方が有利である事、設計や実装が妥当であるとの結論を得た。また、これまで拡張性が高いといわれていたマイクロカーネルも、通信メディア選択機構のような拡張にはあまり役立っていないと思われる。

今後は通信メディア選択機構が複数の通信メディアを同時に使用可能な時に最適な通信メディアを選択するためのポリシーの検討、そのための処理の追加を行なう予定である。バンド幅などの通信メディア間の特性の差は大きい、これに対してはサービスプロキシ [7] と組み合わせることで対処することを検討している。

参考文献

- [1] Mary G. Baker, Xinhua Zhao, Stuart Cheshire, Jonathan Stone, "Supporting Mobility in MosquitoNet", Proceedings of the 1996 USENIX Technical Conference, San Diego, CA, January, 1996.
- [2] C. Perkins, "IP Mobility Support", RFC2002, October, 1996.
- [3] F. Teraoka, Y. Yokote and M. Tokoro "A Network Architecture Providing Host Migration Transparency", ACM SIGCOMM'91, 1991.
- [4] R. Drome, "Dynamic Host Configuration Protocol", RFC1541, October, 1993.
- [5] T. Tokuda, T. Nakajima, and P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System", Proceedings of the 1st USENIX Mach Symposium, October, 1990.
- [6] J. Helander, "Unix under Mach: The Lites Server", Helsinki University of Technology, Master's Thesis, 1994.
- [7] Akihiro Hokimoto, Kuniaki Kurihara, Tatsuo Nakajima, "An Approach for Constructing Mobile Applications using Service Proxies", The 16th International Conference on Distributed Computing System (ICDCS'96), March, 1997.
- [8] Tatsuo Nakajima, Hiroyuki Aizu, Masaru Kobayashi, Kenji Shimamoto, "Environment Server: A System Support for Adaptive Distributed Application", The 2nd International Conference on Worldwide Computing and Its Applications'98 (WWCA'98), March, 1998.