B-37　組込みシステム向けTCP/IPプロトコルスタックの設計

A design of TCP/IP protocol stack for embedded systems

大塚 雄三†　並木 美太郎†
Yuzo Otsuka　Mitaro Namiki

1．はじめに

近年，プリンタやコピー機などLAN接続されてPC等と通信する機器や，情報家電をネットワークで結び，ホームネットワークを構築するようになってきた．そこで，組込みシステムの分野においてもTCP/IPプロトコルによる通信を提供する必要が高まっている．

本論文では，ハードウェア資源に対する制約が厳しいという組込みシステムの特性を考慮したTCP/IPプロトコルスタックの設計について述べる．

2．従来のプロトコルスタックの問題点

従来のプロトコルスタックの処理手順は，まず，受信したパケットをデバイスからプロトコルスタックへと渡す．プロトコルスタックでは，そのパケットのヘッダを解析し，行う処理を決定し，多くの場合はさらに上位のプロトコル，またはアプリケーションにデータをコピーする．

送信の際は，まずアプリケーションなどで作成したデータをプロトコルスタックのバッファへコピーする．そして，そのコピーされたデータにヘッダなどを付与し，パケットとして完成させ，デバイスへ渡して送信する．

この結果，従来のプロトコルスタックではアプリケーションとプロトコルスタック間で少なくとも1回のコピーが必ず起こる．ここで，デバイスがイーサネットの場合には，1つのパケットを送信するのに，最大で1460Byteのデータコピーが起こる．このコピーがなくなれば，その約1.5kBのメモリコピーによるCPU処理や，バッファのためのメモリを節約できる．

3．目標

TCP/IPのプロトコルスタックとそのAPIとして，BSDソケットが有名であるが，データコピーの回数が少なくない，バッファ用のメモリを多用するなどのハードウェア資源について考慮されておらず，組込みシステムには向かないという欠点がある．

そこで，本論文ではアプリケーションとプロトコルスタック間のデータコピーをなくし，ゼロコピーで動作するという特徴をもつプロトコルスタックを設計する．ただし，ゼロコピーで動作するには，ハードウェアがDMAをサポートしているという条件付である．DMAをサポートしておらず，デバイスからの転送時にコピーが起こる場合には，ハードウェアの仕様上回避できないため，一回コピーが起き，1コピーとして動作する．

4．本プロトコルスタックの全体構成

本プロトコルスタックは，図1に示したようにユーザインタフェース部，バッファ管理部，プロトコル部，送受信部からなる．

また，従来のプロトコルスタックでは，プロトコルスタック内部に送受信のためのバッファを持ち，アプリケーションでもバッファを用意していた．しかし，本プロトコルスタックでは，送信用バッファはアプリケーションにのみ存在し，プロトコルスタックでは持っておらず，受信用のバッファはプロトコルスタックにしか存在しない．

送信用のバッファをアプリケーションに持たせた理由は，それぞれのアプリケーションが必要なサイズをその設計時に見積もって確保することで，適切なサイズで確保でき，プロトコルスタックのバッファ領域を減らせると考えた．

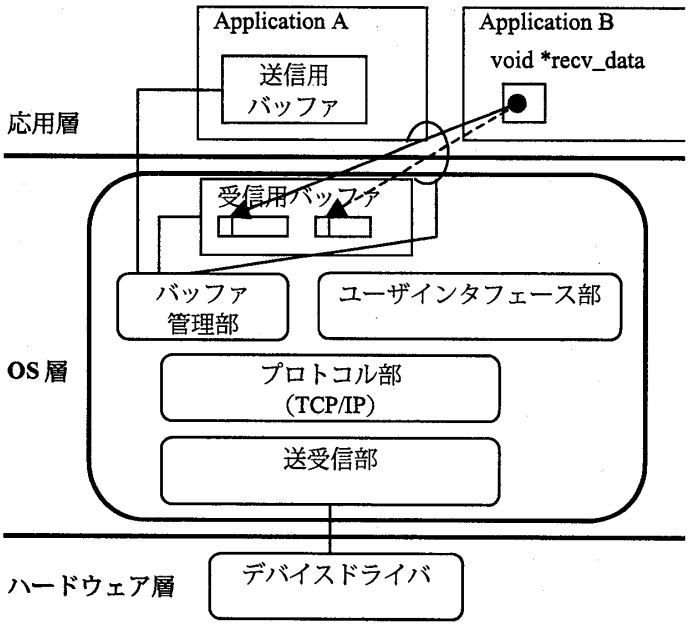

図1　全体構成図

5．プロトコルスタックの処理

5.1．受信の手順

パケットが受信されるとデバイスを通して，プロトコルスタック中のバッファへパケットが格納される．プロトコルスタックではヘッダを解析し，それに従って処理を進める．多くの場合は，さらに上のプロトコル，アプリケーションにデータに渡すことになる．

ここで，このデータを渡す方法として，本プロトコルスタックではデータをコピーするのではなく，データはプロトコルスタック内部に留め，アプリケーションにはそのプロトコルスタック中のバッファにあるデータの先頭アドレスを渡す．そして，アプリケーションはプロトコルスタック中のデータを直接参照する．（次頁の図2）

ここで，プロトコルスタック内のデータを破棄するタイミングについて考える．従来のプロトコルスタックでは，アプリケーションにデータをコピーした時点でプロトコルスタック内から破棄してしまう．しかし，本プロトコルスタックではデータを留めておき，アプリケーションから参照させるので，いつ破棄していいのかを判断することができない．そこで，アプリケーションにデータの読み出しが終わったら，プロトコルスタックへその旨を通知させる．

† 東京農工大学大学院　工学研究科

プロトコルスタックのバッファ，つまりカーネル空間のメモリ領域に対して，通常のユーザプログラムから直接アクセスするということは，保護の面で問題となる．しかし，組込みシステム向けということで，保護よりも性能を重視することにした．

この方法により，従来では受信の際にイーサの場合で最大 1460Byte のメモリコピーが起きていたが，本方式では，ポインタのコピー，つまり 4Byte 程のコピーで済む．

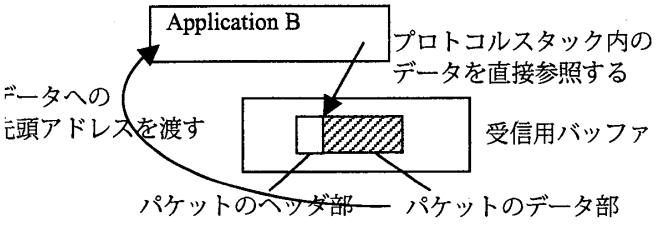

図 2　受信したデータへのアクセス

5.2. 送信の手順

送信のためのバッファ領域は，アプリケーションで確保し，それをプロトコルスタックに登録する．以降，この送信用のバッファはプロトコルスタックで管理するので，その中がどうなっているかはアプリケーションプログラマは気にする必要はない．この理由は次の 5.3 で述べる．

次に，アプリケーションがデータを書き込む．その際にこの送信用のバッファのどこから書き込むべきかを，プロトコルスタックに尋ねる．プロトコルスタックから示された書き込む先頭アドレス，空き領域のサイズに従って，データを書き込む．データの書き込みが終わったら，先頭アドレスと書き込んだサイズをプロトコルスタックに通知し，送信してもらう．プロトコルスタックでは，そのデータにヘッダを付与し，パケットとして完成させ，デバイスへ渡して送信する．

この方法により，送信の場合も受信の場合同様，データコピーが起こらないので，従来ではイーサの場合で最大 1460Byte のメモリコピーが，本方式では，4Byte 程のメモリコピーで済む．

5.3. カプセル化に伴うバッファ中のヘッダフッタ処理

本プロトコルスタックでは，データコピーを行わないので，このアプリケーションで確保されたバッファから直接デバイスへ転送し送信する．デバイスへの転送方法はハードウェアに依存するが，転送元のパケットは連続した領域に格納されていることが望ましい．ここで，送信したいデータは前後に色々なヘッダやフッタがつく，つまりカプセル化される．具体的な例として，送信するパケットが TCP のプロトコルでデバイスがイーサの場合には，送信したいデータの前には 20Byte の TCP ヘッダが付くことによってカプセル化され TCP パケットとなり，その TCP パケットは 20Byte の IP のヘッダが付き，さらにその IP パケットが 14Byte のイーサのヘッダによってカプセル化されイーサのパケット(フレーム)となる．つまり，もともと送信したかったデータの前には，54Byte(20+20+14)ものヘッダがつけられる．最終的にデバイスへ転送するイーサのパケットを連続した領域に書き込むことを考えると，空き領域の先頭からヘッダのために 54Byte 進めた場所からデータを書き込む必要がある．この 54Byte というのはプロトコルやデバイスによって変わる．

受信の際も同様の理由により，プロトコルスタックにある受信用バッファ中にはヘッダフッタそしてデータが，パケットとしての構造で格納されている．

このヘッダなどをアプリケーションプログラマが意識してデータの読み書きすると負担が大きくなる．そこで，アプリケーションには，バッファ中のヘッダやフッタなどの領域は取り除いたデータへのアドレスを示す．特に，送信用のバッファはアプリケーション領域にあるのだが，アプリケーションが自ら場所を指定して書き込むことはない．

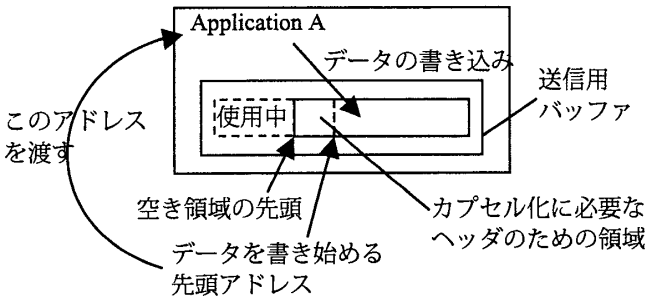

図 3　送信バッファへの書き込み

6．API

本プロトコルスタックでは，以下に示した API を提供している．

TypeSOCK *sk =
cre_sock(char *addr, char *port, char *protocol, int opt);
説明：ソケットを生成する

void **del_sock**(TypeSOCK *sock);
説明：ソケットを削除する

int len = **get_rbuf**(void **p);
説明：受信したパケットのデータのアドレスとサイズを得る．p にアドレスを入れて返される．

void **read_done**(void *p, int len);
説明：データの読み出しの終了を通知する

int err_code = **set_sbuf**(void *buf_addr, int buf_len);
説明：送信用バッファを登録する

int len = **get_sbuf**(void **p);
説明：送信したいデータの書き込みアドレスとサイズを得る．p にアドレスを入れて返される．

int err_code = **send**(void *p, int len, int opt);
説明：送信する

7．おわりに

本論文では，ハードウェア資源に対する制約が厳しいという組込みシステムの特性を考慮に入れ，ゼロコピーで動作するプロトコルスタックの設計を示した．

本プロトコルスタックは，組み込み用 OS「開聞」の上で動作することを想定して設計した．

参考文献

[1] Stephen T. Satchell H.B.J.Clifford 小嶋隆一訳
Linux IP Stacks Commentary，セレンディップ，2001.

[2] 高田広章，組み込みシステムに適した TCP/IP API，
コンピュータシステム・シンポジウム平成 10 年 11 月，
情報処理学会

[3] 萱嶋志門，組込み用 OS の試作，東京農工大学
工学部，1999 年度卒業論文，2000.