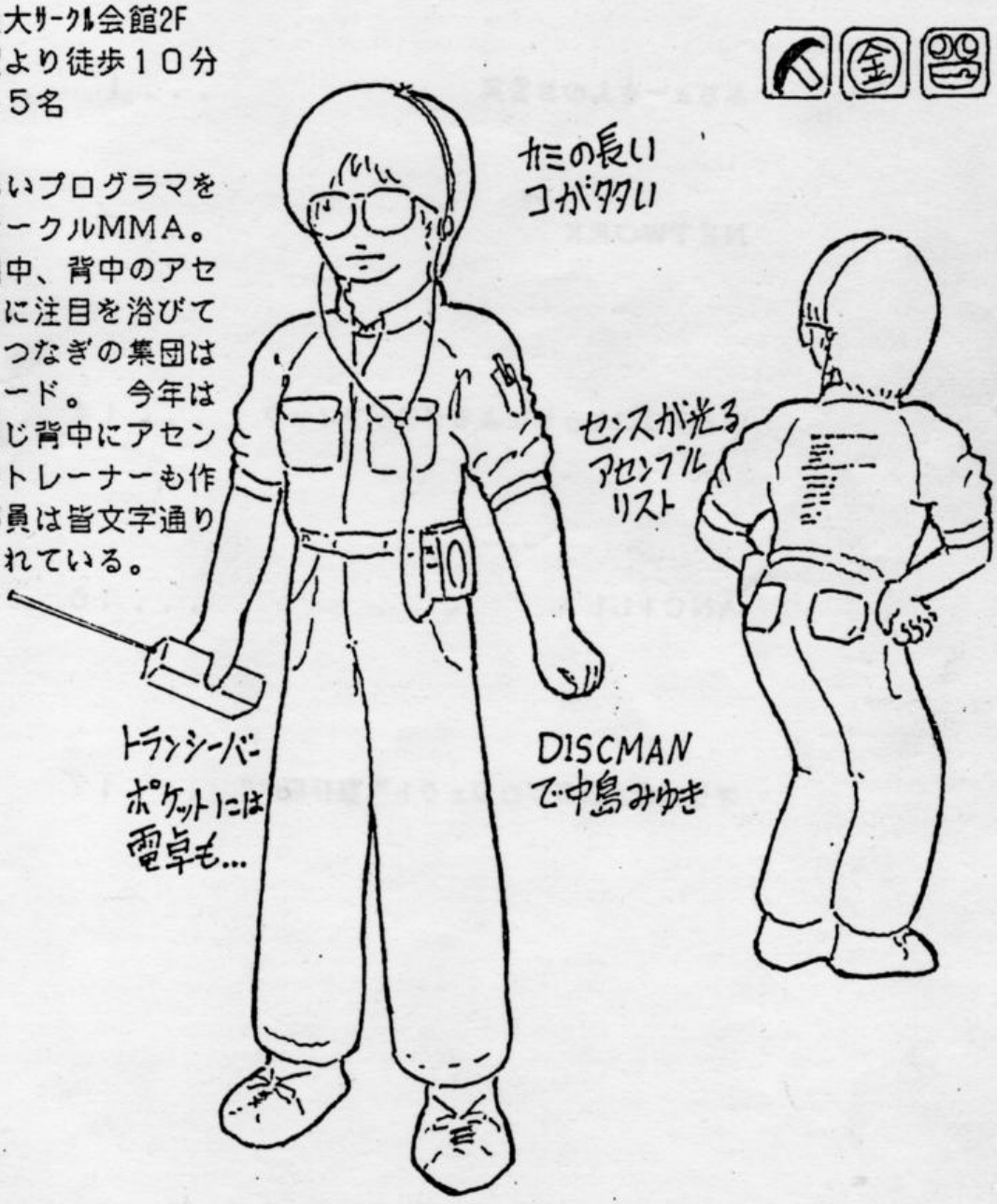
インタラクティブなアーキテクチャをオブジェクト指向で論理設計する

1. マイクロコンピュータを作る会

MMA

[所在地]電通大ｻｰｸﾙ会館2F
[交通]調布駅より徒歩１０分
[部員数]約４５名

[解説]　明るいプログラマを育てる健康サークルMMA。
　調布祭期間中、背中のアセンブルリストに注目を浴びて歩きまわる、つなぎの集団は一種異様なムード。　今年はこの他にも同じ背中にアセンブルリストのトレーナーも作っており、部員は皆文字通り後ろ指をさされている。



[知ってる？]MMAのコンピュータは全部オリジナル。詳細は本文を。

85?

MMA　調布祭　パンフレット　目次

ぶちょーさんのお言葉
―― 今、何故にMMAか。

MMAの設立は１９７６年に遡る。

当時、世間ではまだワンボードのマイクロコンピューターが全盛のころで、いまのような一体型のパーソナルコンピューターなど、ほとんどなかった。ソフトウェアの面でも、ＢＡＳＩＣがようやく出始めたころで、ＯＳといっても、ＣＰ／Ｍがごく一部で使用されているにすぎず、コンパイラなどほとんどなかった。しかし、大型機の世界では事情が違った。大型コンピューターで出来ることが、マイクロコンピューターで出来ない理由はない。少なくとも最低限度のＯＳと、その上で動く、エディタ、マクロアセンブラ、コンパイラぐらいは、実現できるはずだ。売っていないなら、自分たちで作ろう。

当時、ＭＭＡを設立した人々は、このように考えたと伝えられる。

こうして、ＭＭＡは出来た。

「買えないから作る」は、裏返せば「買えるものを買ってしまおう。」でもある。
一通りの勉強、実験を終えた彼らは、当時市販されていたＣＰＵボードを利用し、これを拡張することでシステムを構築していった。

これが後に「ＭＡＩＮ　ＳＹＳＴＥＭ」と呼ばれるもので、最終的には自作のＯＳ（Ｍ／ＤＯＳ）を搭載し、ＰＬＡＮコンパイラ、マクロアセンブラ、リンカ、エディタをはじめとする数々のユーティリティが利用できるシステムへと発展して行くことになる。

しかし、「ＭＡＩＮ　ＳＹＳＴＥＭ」も現在では時代遅れのシステムであり、当然のごとくお蔵入りとなった。そして、それと相前後して、さらに大きなシステムの構築に向けて、新たな歩みを開始していた。

しかし、時代はそれを許さない状況にあった。「手に入らないから作る」で始まったＭＭＡであったが、一昔前の大型機なみのシステムが、今では、非常に安く手にはいる。さらに高級なシステムを目指すには、多くの技術的問題を解決しなければならない。なかでも、最も厄介なのが実装技術である。手配線で作るには余りにも回路が大規模になりすぎた。そのため、ノイズ対策もそろそろ限界に達し、回路が安定に動作しなくなってきている。ソフトウェアについても似たような問題が生じている。

本来、無いものねだり的サークルであるＭＭＡは、今まさに危機に瀕しているといえる。

この調布祭で、限界を迎えつつあるＭＭＡの姿を、見て欲しい。ＭＭＡは、今その限界に挑もうとしている。しかし今のやり方では壁に突き当たるのは見えている。いずれＭＭＡは方向転換を迫られるであろう。

だがＭＭＡに終わりはない。いつの日か、新たに生まれ変わったＭＭＡの姿を見る日がくるだろう。

＜ＮＥＴＷＯＲＫ＞

§１．プロジェクトの目指すもの

現在MMAでは、コンピュータ間に高速の通信手段を提供するネットワークプロジェクトが進行している。このプロジェクトの目指すのは、２～３Ｍｂｐｓの通進路によってコンピュータを接続し、リモートなホスト間での自由なファイルアクセスの手段や、リモートなプロセス間での通信を提供することである。
倶楽部内には数多くのＯＳ９／６８ｋのシステムがあることから、初めはそれらへのインプリメントを行い、徐々に他のホストへのインプリメントをおこなっていく予定である。

そもそも、ネットワークプロジェクトの発足の契機となったのは、異なるＯＳやＣＰＵ間での自由な通信の必要性であった。これを実現するためには、ネットワークのインプリメントは特定のＯＳやＣＰＵに偏ったものであってはならない。我々のアプローチは、次のようになった。

１．　ＯＳＩ７層のモデルでいう、物理レイヤからトランスポートレイヤまでをサポートするノードプロセッサを作製し、各ホストはその機能を利用してネットワークに参加する。
２．　ノードプロセッサとホストとの通信はシステムバス（マルチバス、ＶＭＥバス等）上の共有メモリと、割り込みを通じて実現する。
３．　ひとつのノードプロセッサに複数のホストを接続できるようにし、コストの削減をおこなえるようにする。

上のような方針に従って、現在２枚の、マルチバス上で動作するノードプロセッサが製作されている。この詳細については別の文章で触れることにする。

§2. ノードプロセッサの　ハードウェア

1）特徴

2.5Mｂｉｔ／ｓｅｃの　ネットワーク上のデータ転送速度

改良型トークンパッシング プロトコルを採用

１ネットワーク中に　最大２５５ノードをサポート

ＳＭＣ社ＣＯＭ９０２６を使用

アークネット　コンパチブル

インテル社Ｍｕｌｔｉ－Ｂｕｓインターフェースをサポート

殆どのＭｕｌｔｉ－Ｂｕｓマスターモジュールが使用できる

ノードプロセッサは　モトローラ社ＭＣ６８０００ＭＰＵを採用

2）ラインプロトコル

ラインプロトコルはそれぞれのバイトがスタートインターバルとストップインターバルで区切られているので、独立している。それぞれのデータバイトは一定の時間のインターバルを持ち１ｂｙｔｅの転送に４．４マイクロセコンドかかる。

送信がアラートバーストで始まり、８ｂｉｔのデータキャラクターが２ユニットのマークと１ユニットのインターバルをおいて送られる。以下に示すように、５種類の送信形式がある。

１．送信開始

アラートバーストに続いて３つのキャラクターが送られる。ＥＯＴと２つのＤＩＤキャラクターである。このメッセージは１つのノードから他のノードへトークンを送るために用いられる。

（注）ＤＩＤ　送信先識別子

２．空バッファ調査

アラートバーストに続いて３つのキャラクターが送られる。３つのキャラクターはＥＮＱと２つのＤＩＤである。このメッセージは他のノードにデータパケットを受け取る事が出来るかどうかを尋ねるために用いられる。

３．データパケット

アラートバーストに続いて以下のキャラクターが送られる。

- ＳＯＨ　(START OF HEADER)
- ＳＩＤ　(SOURCE IDENTIFICATION)
- ＤＩＤ　２回
- パケット長
- ＣＲＣ

4．送信許可応答

アラートバーストに続いて１つのキャラクターが送られる。

１つのキャラクターはＡＣＫである。（２）の答えとして空バッファがあるときに送られる。

5．送信不許可応答

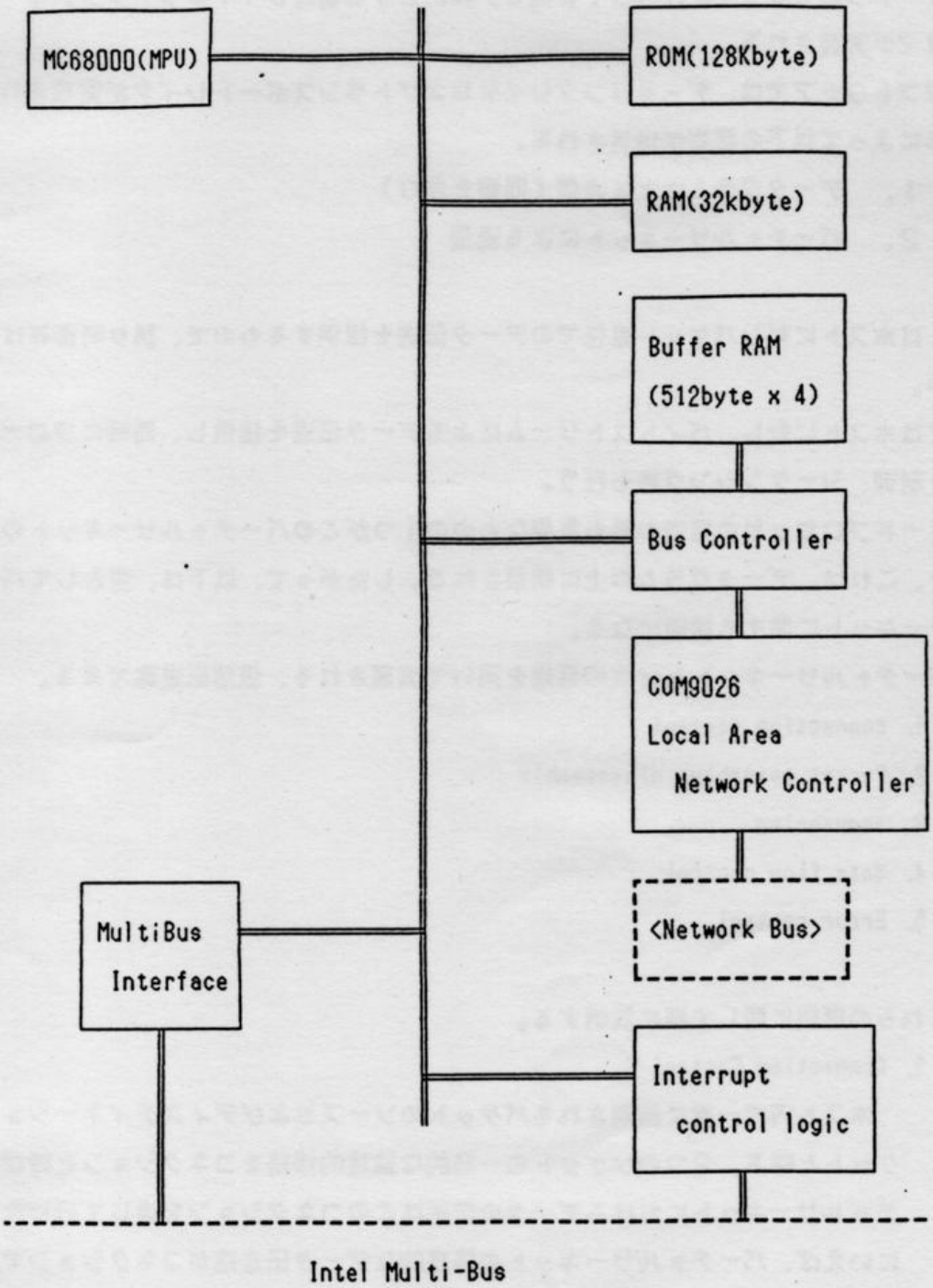
アラートバーストに続いて１つのキャラクターが送られる。

１つのキャラクターとはＮＡＫである。（２）の答えとして空バッファが無いときに送られる。

3）ネットワークプロトコル

ネットワーク上のプロトコルは”改良型トークンパッシングプロトコルに基いている。改良型トークンパッシングとはトークンをうけとったノードが応答を返すためである。ネットワークの構成と管理がＣＯＭ９０２６のマイクロプログラムシーケンサーによって行われる。プロセッサーやインテリジェントペリフェラルは送りたいデータパケットとディスティネイションアイディーを空バッファに送るだけで良い。するとＣＯＭ９０２６はつぎのトークンを受け取ったときに空バッファ調査を行い受信ノードが送信許可応答を返してきたときに送信される。パケットの受信が成功したならば受信ノードは（４）の応答をして送信が終了する。

4）構成図


MC68000(MPU)
ROM(128Kbyte)
RAM(32kbyte)
Buffer RAM
(512byte x 4)
Bus Controller
COM9026
Local Area
Network Controller
<Network Bus>
MultiBus
Interface
Interrupt
control logic
Intel Multi-Bus

§３．ノードプロセッサの提供するソフトウェアやモデル、サービスの種類について

ノードプロセッサ上にはＯＳＩ参照モデルにおける物理レイヤからトランスポートレイヤまでが実装される。

ソフトウェアでは、データリンクレイヤおよびトランスポートレイヤが実現される。これらによって以下の機能が提供される。

１．　データグラムによる通信（同報を含む）

２．　バーチャルサーキットによる通信

１はホストに対しパケット単位でのデータ伝送を提供するもので、誤り制御等は行われない。

２はホストに対し、バイトストリームによるデータ伝送を提供し、同時にフロー制御、誤り制御、シーケンシング等も行う。

ノードプロセッサの機能の最も重要なものの１つがこのバーチャルサーキットの提供であり、これは、データグラムの上に構築される。したがって、以下は、主としてバーチャルサーキットに関する説明になる。

バーチャルサーキットとは次の機能を用いて実現される、仮想伝送路である。

1. connection control
2. Packet assembly, disassembly
3. sequencing
4. data flow control
5. Error control

これらの機能に関して順に説明する。

1. Connection Control

　ホスト内で一意に識別されるパケットのソースおよびディステイネーションをソケットと呼ぶ。２つのソケットの一時的な論理的接続をコネクションと呼び、バーチャルサーキットにおけるデータの伝送はこのコネクションを通じて行われる。逆にいえば、バーチャルサーキットの論理的なデータ伝走路がコネクションであり、その両端がソケットである。バーチャルサーキットにおける通信はまずコネクションの確立に始まり、以後、このコネクションを通じ以下に述べる機能を用いて伝送されるデータの正当性を保証することにより行われる。さらに、広域のネットワークにおいてはルーティング（複数の伝送路が存在する場合に、どの経路を選択するか）の問題が生ずるが、本ネットワークにおいては、ルーティングに対する拡張性を確保し、他のネットワークと接続する場合は、ゲートウェイ（他のネットワーク

とのインターフェースをとる部分）においてルーティングを行うものとする。

2. Packet Assembly Disassembly

PAD と略されるこの機能は、ホストから渡されるデータストリームを、実際の伝送路が受け付けられるサイズに区切り、伝送先アドレスシーケンス番号をふくむヘッダ情報を付け加えパケットストリームとしてホストへ受け渡すものである。

3. Sequencing

バイトストリームをパケッティングして、転送すると、パケットの順番が狂うことがある。これを防ぐのがsequencingで、パケッティングをする際に、各パケットにシーケンス番号をふり、受信側でそれをチェックすることにより受信パケットの順序の正当化を保証する事を目的とするものである。

4. Data Flow Control

これは効率よくデータを伝送するためにデータの流れを制御するもので、相手のアクノリッジを待たずに送出できるパケット数（これをウィンドウサイズという）をコネクションの開通時に決定し、以後、一括アクノリッジによるパケット交換を行うことにより伝送効率の向上がはかられる。固定ウィンドウサイズ、可変ウィンドウサイズの2つの方式があるが、このネットワークでは、固定ウィンドウサイズを用いた特殊なスライディングウィンドウアルゴリズムを用いる予定である。

5. Error Control

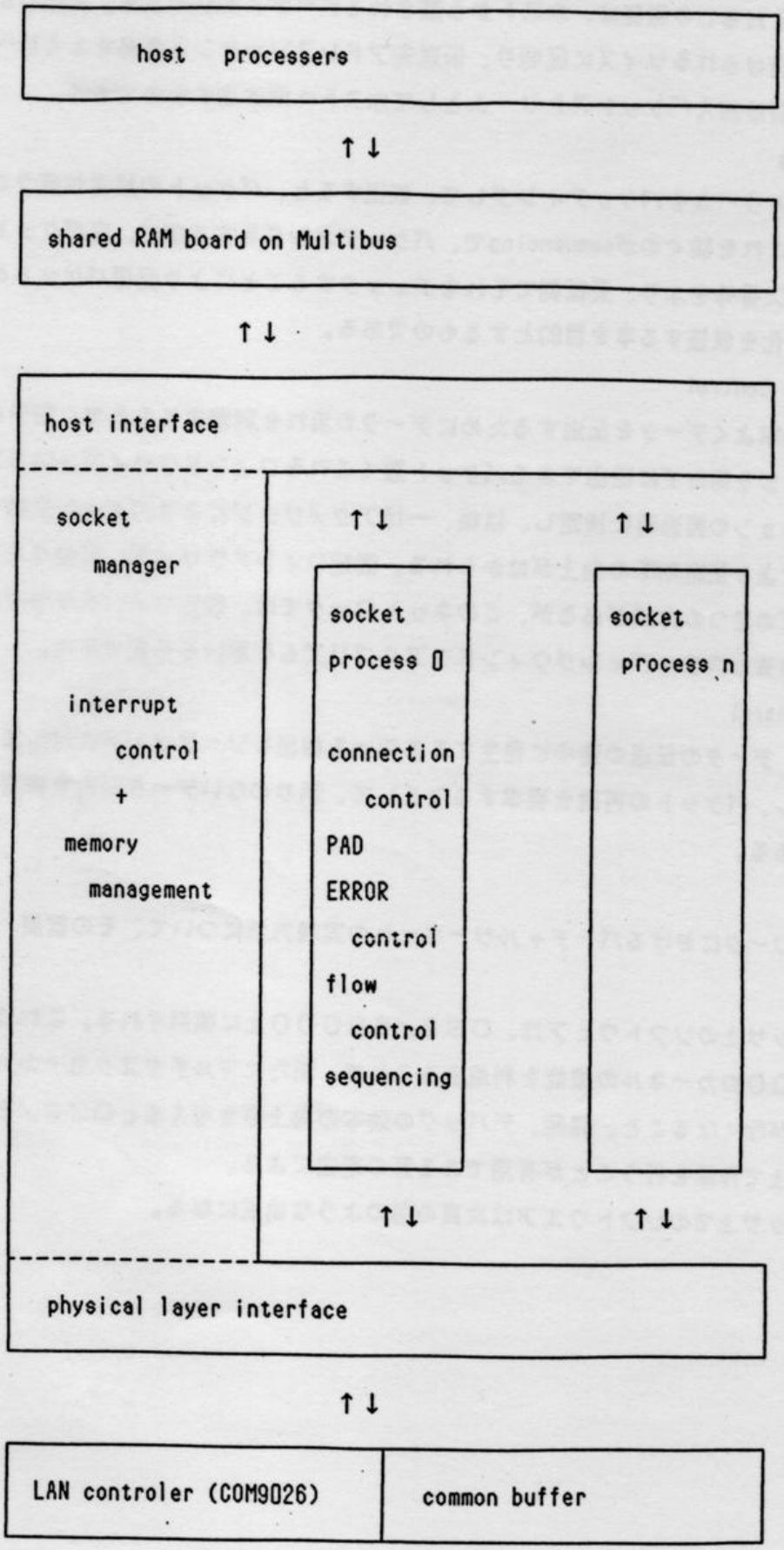
これは、データの伝送の途中に発生するエラーを検出しソースノードに対しエラーを報告し、パケットの再送を要求するなどして、誤りのないデータ伝送を保証するものである。

次に本ネットワークにおけるバーチャルサーキットの実現方法について、その概要　を述べる。

ノードプロセッサ上のソフトウェアは、ＯＳ９／６８０００上に構築される。これはＯＳ９／６８０００のカーネルの機能を利用することで、新たにマルチタスクカーネルを用意する必要がなくなること、開発、デバッグの効率の向上等を考えるとＯＳ９／６８０００の環境上で作業を行うことが有用である等の理由による。

ノードプロセッサ上でのソフトウエアは次頁の図のような構成になる。

ノードプロセッサ　ソフトウェア構成図



↑↓はその間でデータのやり取りないしは、プロセス間の通信が行われることを示す。また、点線部は本来独立しているべきであるが、効率の関係から、1つのモジュールとなっていることを示している。

この図からわかるように、ノードプロセッサ上のソフトウエアは次の4つに分かれる。

1. Host Interface
2. Phisical Layer Interface
3. Socket Manager
4. Socket Driver

それぞれの機能は、以下の通りである。

1. Host Interface

   マルチバス上の共有RAMを通じて、ホストとの間の通信を行う部分で、特定領域へのコマンドのセットおよびインタラプトの発行により通信を行う。

2. Physical Layer Interface

   ノードプロセッサ上のハードウェアを直接操作する部分

3. Socket Manager

   socket driver processの生成、共有RAM上のメモリの管理等を行う。またinterrupt handler routineもここに含まれている。

4. Socket driver

   socket managerによりforkされるプロセスでソッケト1つに対して1つのプロセスが対応する。PAD、Error Control, Multibus上の共有RAMを直接アクセスする場合がある。

以上の構成の基に、ノードプロセッサはホストに対し次に示すようなサービスを提供する。

1. socket no_reg

   ソケットナンバーの割り当ての要求

2. clear_socket_no

   ソケットナンバーの返還

3. connection_open

   2つのソケット間におけるコネクションの確立要求

4. connection_close

   確立されているコネクションの解放

5. send

データの送出

6. recieve

データの受信

7. abort

コネクションの中断

8. send_to

データグラムによる、パケットの送出

9. recieve_from

データグラムによるパケットの受信

§４．ユーザに提供されるネットワークライブラリについて

１）はじめに

ユーザプログラムに対してネットワークの機能をどの様な形で提供するかという点について、我々は２つの方針があると考えた。ひとつはＵＮＩＸ　４．２ＢＳＤの、クライアント←→サーバーモデルであり、いまひとつはＯＳ９のネットワーク（マイクロウエアが提供している）のデバイスオリエンテッドなモデルである。それぞれがどのようなものかを、簡単に説明しよう。

１．ＵＮＩＸ　ＢＳＤ　４．２　のアプローチ

ユーザに提供されるのは、ＯＳＩのモデルと良く似通っている。すなわち、ＯＳＩモデルのデータグラム、バーチャルサーキット（用語解説参照）を使うためのＣのライブラリを用意している。ユーザは、これらを利用して、リモートファイルトランスファー、リモートプリンタサーバ、リモートログインサーバ等のアプリケーションを作成する。とは言っても、直接にこれらのアプリケーションを作成するユーザは限られたシステムプログラマであり、一般のユーザは、これらのアプリケーションプログラムを利用するに留まるのが実状であろう。

２．ＯＳ９のアプローチ

ＯＳ９上では、ネットワークは１つの入出力デバイスとして扱うことができ、リモートなシステム上のパスを指定するには、次のようなパスリストを使う。

／ネットワーク名／ホストの名前／パスリスト

例えば、次のように使うことができる。

```
copy  /net1/Ares/h0/foo  /Ether/Tess/d0/user/kei/bar
```

これは、ｎｅｔ１というネットワーク上の、ホストネームＡｒｅｓの／ｈ０／ｆｏｏというファイルを、ネットワークＥｔｈｅｒ上のホストネームＴｅｓｓの／ｄ０／ｕｓｅｒ／ｋｅｉ／ｂａｒにコピーする。

また、

```
list  /d0/user/kei/bar  >/net1/Ares/p
```

は、／ｄ０／ｕｓｅｒ／ｋｅｉ／ｂａｒなるファイルを、ｎｅｔ１上のホストＡｒｅｓに接続されているプリンタ（デバイス名ｐ）に出力（リダイレクト）する。これによりリモートなプリンタへのリスティングを行うことができる。

このように、ＯＳ９はファイルやデバイスを、リモートなものかどうかにかかわらずトランスペアレントに利用できる。これが我々がＯＳ９のネットワークをデバイス指向と呼ぶ理由である。

それぞれのアプローチの概要は以上の通りだが、結論を先に言えば、我々はＢＳＤ４．２のアプローチをとった。ＯＳ９のアプローチには汎用性において問題が認められたためである。次のような例を考えてみて欲しい。

リモートな、複数ホストからのリストアウト要求にも応えることのできるプリンタサーバを設計したいとする。この場合、ＢＳＤのアプローチに従えば、バーチュアルサーキットをつかってこともなげに記述する事ができるが、ＯＳ９のアプローチではどうすればよいというのだろうか？恐らくプリンタサーバ用の仮想的なデバイスを作成し、それらへリモート・リダイレクトすることによって実現するしかないだろう。こうすると、新たなサーバ（ログインサーバ、等々）を作る度に、新たなデバイスドライバを設計しなければならない。しかしながらＯＳ９のデバイスドライバは、その詳細な情報が提供されておらず、まったく新しいデバイスドライバの作製は困難を極める。

相互の得失を見きわめた上で、我々はＢＳＤのアプローチをとることに決定した。

2）ネットワーク機能ライブラリの概要

前節で述べたように我々はＢＳＤ４．２と同等のアプローチをとることにしたが、実際にネットワークの機能を利用する際に、どのようなライブラリ関数をどのように使って処理を進めるのかを、具体例を示しながら説明したい。ここでは、プリンタサーバとそのクライアント（リモートホストのサーバと通信し、リストアウトを要求するプロセス）を例にとって説明する。

まず、クライアントはリモートプリントサービスを行っているサーバプロセスのネットワークアドレス（以降ＮＡＤ）を知らなければならない。このために、getnatbyname()という関数が用意されている。この関数は各ホスト中にローカルに存在するサービスネームテーブルを検索して、そのサービスを提供しているサーバプロセスのＮＡＤを返す。このサービスネームテーブルはサーバホストのダウンや、起動（電源ＯＮ等）に応じて、ダイナミックにアップデートされる。

次に、得られたＮＡＤへ対してcon_req()を使ってコネクションの確立を要求する。

サーバプロセスは、accept()を呼び出すことによって、どこかからのコネクション要求を待っているが、要求が到着するとその要求を行ったプロセスとの間にコネクションを確立しそのコネクションに対してサービスするプロセスを起動する。以降、そのプロセスがクライアントから送られてくるリストアウトファイルをスプーラー用のディレクトリにコピーする。

クライアントは正常にコネクションが確立されたら、そのコネクションを通じてリストアウトの方法（高密度印字やレタークオリティ、希望するプリンタの種類等々）、ファイル名、ファイルの本体等をsend()を使って送り、最後にコネクションをcon_close()を使って終結させる。

プリントサービスの概要は上のようになるが、上に現れたライブラリ関数以外にも多くの関数が存在する。例えば例中にも示された、サービスネームテーブルのアップデートには、一斉同報機能を利用するのが妥当であるが、このためにはバーチュアルサーキットではなく、データグラムをサポートする関数が、ライブラリ中に存在する。

§５．ネットワーク用アプリケーションについて

ネットワークによって結合されたシステムに対して、ユーザはどのような機能を要求するだろうか。我々がネットワークを構築する契機となったのは、リモートなファイル転送や、プリンタサーバの必要性であった。これらの機能を提供するためのネットワークユーティリティは早急に準備されなければならない。

また、ＢＳＤ４．２のアプローチをとることによって失われた、デバイスへのトランスペアレンシーを取り戻すための、ネットワーク用ｓｈｅｌｌ（ｎｓｈ）を作製する事も必要であろう。ここでは、我々のネットワークの将来像を示す意味でも、このｎｓｈが、どのようなものになるのかを考えてみたい。

１．　コマンドラインによって与えられたパスリストを解析し、それがリモートなホスト上へのパスであった場合には、然るべきホストのサーバと通信を行って、コマンドを処理する。例えば、

```
list   /net/Ares/h0/foo   >/net/Tess/d0/bar
```

というコマンドラインが投入された場合、ホストネームＡｒｅｓ上のファイルサーバに対して／ｈ０／ｆｏｏなるファイルの転送を要求し、それをｌｉｓｔコマンドへ渡してやる。また、リモートホストへのリダイレクトが指定されているので、ｎｓｈは、ｌｉｓｔをｆｏｒｋする前に、ｓｔｄｏｕｔをテンポラリファイルにリダイレクトしておき、ｌｉｓｔプロセスの終了後、テンポラリファイルを、Ｔｅｓｓ上のファイルサーバと通信を行いながら／ｄ０／ｂａｒへとコピーする。

２．　ネットワーク全体でのｃｐｕに負荷分散を行う。ネットワーク上に複数のｃｐｕ（ホスト）が存在するのだから、そのうちのロードアベレージの低いホスト上でコマンドの実行を行う事が考えられる。とは言っても、ｉ／ｏバウンドのジョブを他ホストで実行しても意味はないため、そのコマンドの性格を、ｃｐｕ／ｉ／ｏレシオによって把握する必要ががある。このための、コマンドごとのデータベースが必要となるだろう。または、リモート実行については、ユーザ（コマンドラインの投入者）が指定を行うという方針も考えられる。例えば次の様になる。

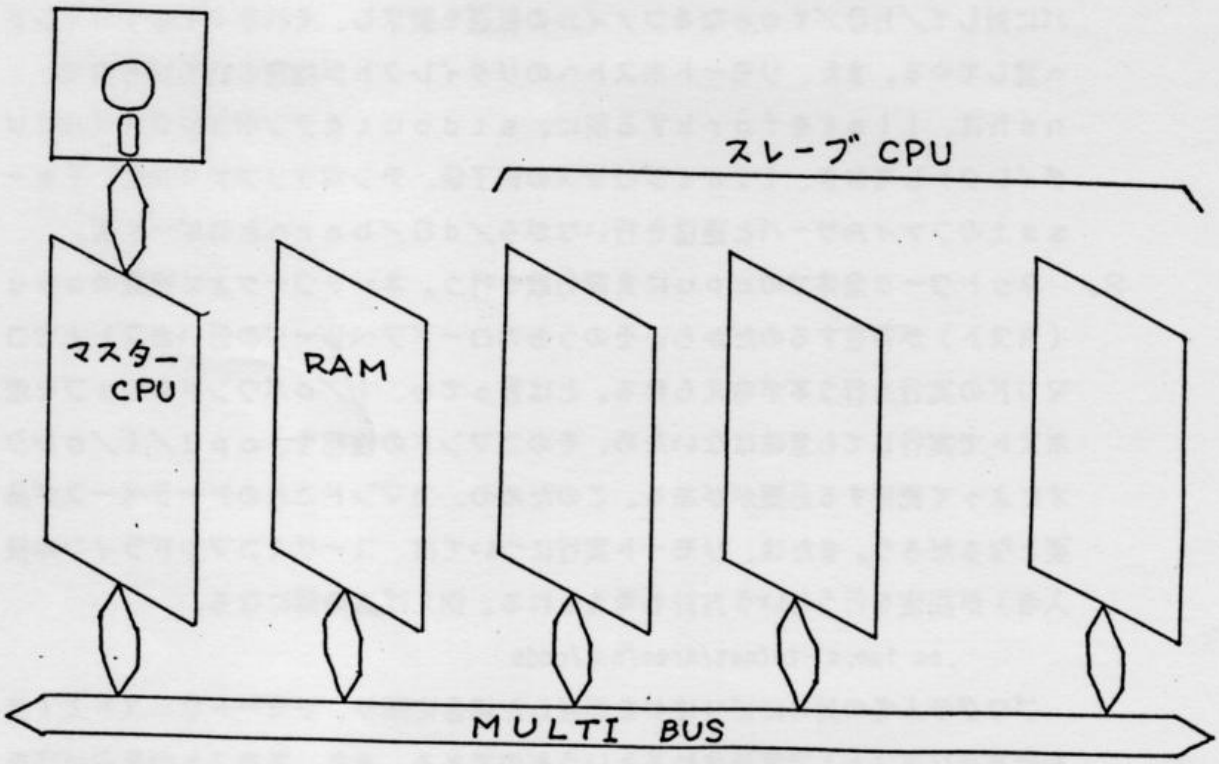
```
.cc foo.c -f=/net/Ares/hø/cmds
```

プログラム名の始めにピリオドを前置した場合に限り、リモートなホスト上（最も軽負荷なホスト）で実行されるというものである。また、各ホストの負荷状態を知るための、ダイナミックに更新されうるデータベースを用意する必要がある。この更新のためには、データグラムプロトコルの一斉同報機能を利用するのが適切であろう。

＜マルチプロセッサによるグラフィック＞

我がMMAでは２年前より、自作フレームバッファと、レイトレーシングを用いて３次元物体の表示の実験を行ってきました。

このレイトレーシングというアルゴリズムは、マルチプロセッサに向いています。そこで、68000 マイクロプロセッサを複数使ってこれを行ってみようと作業中です。それぞれの 68000 はマルチバスのボードに乗っています。このボードには cpu 周辺の回路のほかに 128KB のメモリや I/O も乗っていて、ディスクインターフェースを追加すれば簡単に os9/68000 を走らせることができます。このボードを何枚かとマルチバスからアクセスできるメモリボードを１つのラックに差して１つのシステムとします。各 cpu ボード間の通信はメモリボードを使って行うことにします。



（図・システム構成）

システムの中に１つだけあるマスター cpu にはフロッピーディスクインターフェースがついていて、まずこのボードで os9 を立ち上げ、そのあとメモリボードを RAM ディスクとしてイニシャライズします。フロッピーディスクのついていないスレーブ cpu は、この RAM ディスクから os9 を立ち上げます。

このシステムを構築するためには os9 のデバイスドライバを書かなければなりません。os9 のデバイスドライバはアセンブラを使って書く事ができますが、できれば、Cなどの比較的高級な言語を使ってドキュメント性を上げたいところです。しかし、Microware からは、Cでデバイスドライバを書くためのプログラム環境は、供給されていません。そこでこの環境を自力で構築することにしました。

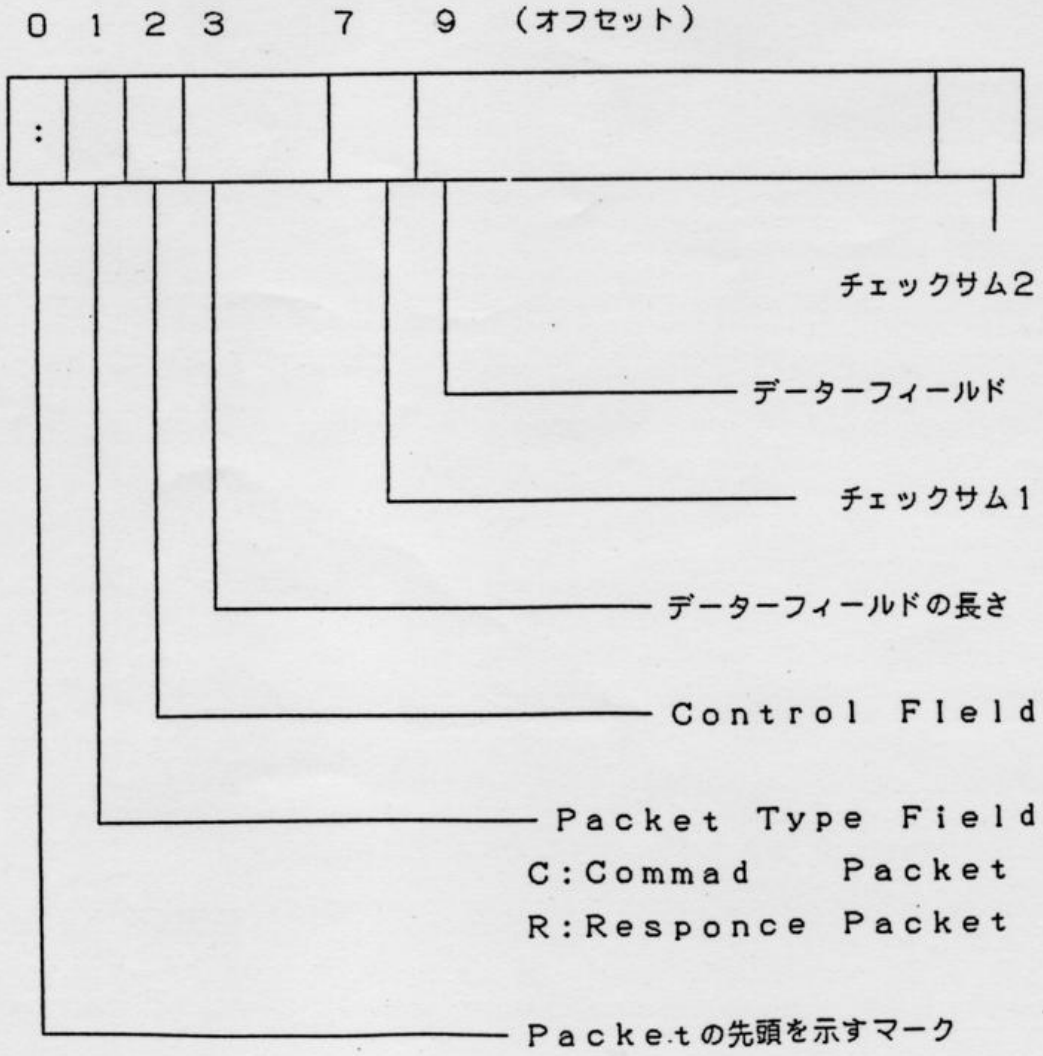
このシステムが完成すれば、cpu パワーは増大し、より高度なテクニックを使った実験も短時間で行う事ができるようになるでしょう。

<ANCILLA>

現在我が部で保有するＩ／Ｏ機器の種類および量の増加によりそれらの統一的なアクセスが望まれている。ＡＮＣＩＬＬＡはＭＳＣＳ（MMA Serial Communicator Standard）を通じて仮想Ｉ／Ｏ化された各種デバイス（ＲＯＭシミュレーター，ｐｌｕｍ，ＲＯＭライター　ｅｔｃ．）を利用できるようにするためのものである。この仮想化されたＩ／Ｏの利点は現存するデバイスをバージョンアップしたり、新しく作り直したとしても相違点をＡＮＣＩＬＬＡが吸収するため、ホスト側では何ら手を加える必要が無くなる。

ＡＮＣＩＬＬＡ本体がサポートする部分はホスト側とのやり取りと直接デバイスをアクセスするＰＩＯ（パラレル・インターフェース）をコントロールするための関数群である。ホスト側とはＭＳＣＳを通じてパケット通信を行う。パケットのフォーマットを示す。本体に加え各デバイスごとのサブルーチンを、リンクして使用する。

ハードウエアはＺ－８０をＣＰＵとしてＲＯＭ　８Ｋｂｙｔｅ，ＲＡＭ　２Ｋｂｙｔｅ，ＰＩＯ，ＳＩＯ，ＣＴＣ　等を載せている。



オリジナル ＯＳ プロジェクト "聖杯探求"

by　Meta

かつてホビーストが初めてマイクロプロセッサを手に入れた頃、マイコンと言えば８ｂｉｔのＣＰＵ，２５６ｂｙｔｅのＲＡＭ、３０個程度のスイッチが付いたパネル、８桁の７セグＬＥＤと言った程度の物であり、大抵の人々の最高目標と言えば８ＫＢのメモリとターミナルを接続してかの"STAR TREK"を動かす事であった。ホビーストの大部分がハードウェアから自作していた時代であり、コンピュータの機能こそ低かったものの、オーナーやユーザーがシステムの全機能を理解し、管理していた時代であった。

時は流れ、今やマイコンはパーコン（もしくはパソコン）と名を変えた。現時点に於ての標準的なシステムは１６ｂｉｔＣＰＵ，０．５ＭＢ以上のメモリと２０ＭＢ程度のハードディスクを装備しており、特に、ＥＷＳ（エンジニアリングワークステーション）と呼ばれる先進的なシステムに於ては、先に述べた様なハードウェア的な面もさることながら、マン・マシン・インターフェースやＯＳ、ネットワーク機能等、ソフトウェアを含めた「システム」としての機能の高さが問題となっている。

この様に機能が複雑になればなるほど、ＯＳの中枢部として出来合いのソフトウェアを利用することが多くなり、システム全体を把握しているユーザーは少なくなって行く。例えば現在マイコン界で話題となっているＯＳ，Ｕｎｉｘについてもその技術資料が公開されているにも拘らず、その全てを把握しているユーザーはほんの一握りしかいないであろうと推測される。

勿論、全てのユーザーがシステムの全機能を把握している必要はあるまい。だが、我々のようにシステムの完全自作を理想とする集団にとって中身の見えないシステムは、機能の不充分なシステムと同程度に不愉快なものであり、開発途上の一時的な物としてはともかく、恒久的に使用するシステムとしては我慢がならない。

マイクロコンピュータの？明期のように全てがアマチュアの手によって開発された時代ならともかく、現在のように企業の手によって商業的にシステムが開発される時代では他人の開発したソフトウェアの公開を望むのには無理がある。しかし、我々はこの点に於てだけは妥協したくないと思っている。従って他人に頼れないから自分達で作り上げていくしか方法はないであろう。

開発環境を自らが作り出し、その上であらたな環境を構築していく、これが我々の理想とするシステムの状態であり、それを作り出すのが以下に挙げるプロジェクト「聖杯探求」である。

広く知られている通り、聖杯探求とは英国に伝わる伝説の１つであり、一時はその徳をもって数多くの騎士を集め、絶頂期を迎えたアーサー王朝が、聖杯 ― より高き理想 ― を求めた故にその終末への第一歩を踏み始める、という悲劇的な逸話である。我々が望むものも、あるいは伝説と同じ結末を招くのかも知れない。それを知りながらも、我々は聖杯の存在を知る者として、自らの理想を求めて果てのない旅に発つのである。

・聖杯としてのMultics

現在マイコン用のOSとして最も脚光を浴びているシステムといえば、大抵の人はUnixを挙げるに違いあるまい。Unixはよく知られているようにベル研究所の僅か二人の研究者の手によって開発されたミニコン用のOSである。ところで、Unixの特徴として知られている多くの点はUnixの開発者達も参加していたコンピュータユーティリティ構築プロジェクトの成果であるOS, Multics のものであることを御存知であろうか。Multics こそ世界で初めてコンピュータ社会における理想を目指した者達の聖杯なのである。

Multics は1960年にMIT とベル研究所とGE社が共同で開発を始めたOSである。その目標は将来の情報社会におけるコンピュータの共同利用を想定して、安全で快適なコンピュータシステムを構築することにあった。Multics の開発において研究者達は、従来にない画期的なアイデアを次々に実現していった。幾つかそのアイデアを紹介すると、OSの記述に高級言語を用いたことや、ページドセグメントを採用した仮想記憶機構、仮想記憶を前提とした実記憶と補助記憶の一体化。木構造に階層化されたファイルシステム、独立したコマンドインタープリタ等である。これらのアイデアの大部分は現在の大型コンピュータの基本的なアーキテクチャとして取り込まれていることからも、その先進性は想像出来るであろう。

Multics はこのように当時の研究者達の理想を求める心が生んだシステムであった。しかし当時の彼等にはMultics のような巨大システムを開発した経験はなく、開発には非常な困難が伴った。まだ当時の計算機は現在とは比べものにならない程遅く、それに比較してMultics はあまりにも思いシステムであった。

９年の長期を要して完成したシステムは非常に巨大で重く誰にもその全貌が理解できないシステムになってしまっていた。やがてMultics の開発プロジェクトチームは解散しMIT やベル研究所の研究者達は外の研究に移り、GE社はコンピュータ事業から撤退してしまうことになる。その後Multics はGE社のコンピュータ部門を買い取ったハネウェル社によって保守されることになるが、OSの主流になることはなく歴史のなかに消えていこうとしている。世界中のコンピュータ研究者に注目されたMultics プロジェクトはこうして失敗に終わったのである。

ところが、Multics の開発に於いてそのＩ／Ｏ部を担当していたベル研の研究者達は、プロジェクトが失敗に終わったのにも拘らず、その快適なコンピュータ環境を失うことを嫌い、より小さなコストで同様の環境をミニコン上に構築した。現在Unixと呼ばれているOSの元祖である。彼らがコストを抑える為に妥協した点は、Multics の最重要テーマの一つ、「安全性」であった。彼らにとって大切だったことは、Multics と同程度の環境を自分達（トップレベルのコンピューター科学者）が利用できる様にすることであった。従って、Unixはシステムを正しく理解している者達が、他人（やシステム）に対して悪意を持たずにいる場合だけに快適に利用できるOSとなっている。現在のUnixは安全性を改善するために種々の改良が加えられているが、システムの基本設計にその思想が盛り込まれていないために、Unix本来の簡潔性が失われつつある。

その他にも、Multics ではファイルのリンクに関しても斬新な手法が採用されており、設計者達のレベルの高さを伺わせるものとなっているが、Unixではコストの関係からそれらの機能も削除されており、より理想から離れたシステムとなっている。

・聖杯探求計画

我々にとっての理想のOSを開発すべく、ここ数年の間我々は既存のOSを研究してきた。CP/M, MS-DOSは論外であったし、大型機のOSも決して使い易いものではなかった。Unixはかなり使い易いシステムではあったが、ローカルネットワークに接続した場合いくつか問題を含んでいた。OS-9も中々興味深いシステムであったが使用してみるとUnixのサブセットとしてはあまりにも使い勝手が悪いものであった。

現在までの調査でもっとも我々の理想に近いシステムはUnixの4.2BSDとアポロDOMAINのOSであるAGESである。4.2BSDはその豊富なユーティリティ群とソケットを使ったプロセス間通信が興味深く、AGESはネットワーク指向のアーキテクチャ特にプロセスの各ノードへの分配とネットワーク上でトランスペアレントなファイルシステムが我々の目標に合致していた。

我々は現在ネットワークじょうでの分散型計算機システムを開発すべく各プロジェクトを進行中である。ソフトウェアの面では開発に必要なクロス環境すなわちクロスアセンブラ、クロスコンパイラ、それに平行処理に必要なマルチタスクカーネル等を開発中で一部はすでに開発を完了している。ハードウェアの面ではNS32032 CPU にファミリのMMU とFPU を加えたボードにネットワークを実現するためのノードプロセッサ等を開発中である。

OSの詳細な仕様は現在まだ流動的であり、クロス環境が完成した後に決定されるであろう。このプロジェクトはMMA の総決算であり今後の我がクラブの有りかたをも問う重大な計画である。聖杯探求に失敗すれば我々は他のコンピュータサークルのようにゲーム屋に堕落してしまうことになるだろう。神の加護があることを祈るばかりである。

・ハードウェア

クラブに用意されているハードウェア資源としては、NS32016 （16bit バス）CPUを搭載したマルチバスボードが用意されている。これは２年前に試作されたものであり、当時のデリバリの状況からFPU （浮動小数点プロセッサ）以外のファミリーは搭載されていない。割り込みコントローラ等は全てインテルの製品を利用した。

このボード上で、簡易デバッガを始め数種類の評価用プログラムを走らせ、プロセッサの評価をおこなった。

現在では、ＮＳ社からは32bit のバス幅を持つNS32032cpuがリリースされている。又、VAX と同程度の機能を持つMMU も入手可能になっているため、新たに、より大規模なプロセッサボード（バーサスボード）を開発中である。このボード上には、現在入手可能な全てのファミリーチップをはじめ、SCSIデバイス（ハードディスクインターフェース）等のＩ／Ｏも豊富に搭載し、今後２年程度の使用には充分耐える様考慮されている。今後はこのボード上に種々のソフトウェアシステムを構築し、聖杯を探す足掛りとしたいと考えている。

・ソフトウェア

― 簡易デバッガ ―

プロセッサの評価を行う為に、トレースやダウンロード機能を備えた機械語モニタを用意し、ボード上のROM に固定した。プログラムはアセンブラできじゅつした。

― アセンブラ ―

２年前にCP/M上でPascalを用いて記述し利用していたが、変換速度が余りにも遅く、又プログラムのポータビリティにも欠けていたので今年Ｃ言語を用いて全面的に記述しなおした。速度は約５０倍に向上し、又Ｃ言語を利用したため種々のOSへ移植刷ることが出来た。現在のホストマシンは市販のMS-DOSパソコンであるが、近日中にOS-9/68K上に開発環境を移したいとかんがえている。（4000行）

— コンパイラ —

現在、Ada ライクなシンタックスを持ったクロスコンパイラを開発中である。プロトタイプは完成しているが、開発者の時間的余裕・機能追加等の問題があり未だ完成していない（約5000行）。

— リンカ —

32000 プロセッサのソフトウェアモジュール機能をフルに活用したクロスリンカが開発中である。設計は既に終了し、プログラミングを残すのみとなっているが、現在32000 側のソフトウェアシステムが移行期にある（簡易デバッガ→マルチタスクモニタ）ため製作は一時中止されている。

— マルチタスクモニタ —

クラブ内におけるＮＳ３２０３２プロセッサを使った計算環境構築プロジェクト「聖杯探求計画」の一環としてマルチタスクカーネルを製作した。

このマルチタスクカーネルは、我がＭＭＡにおけるＮＳ３２０３２ボード上で動作する予定である自作のオペレーティングシステムの最内層に位置し、マルチプロセスの環境におけるプロセス間の同期、通信の基本的な機能を実現するものである。その機能としては、プロセスキューの管理、プロセスの切り替え機構、割り込みハンドリング機構だけの単純なものではあるが、このマルチタスクカーネルの機能を使うことにより、より高機能なマルチタスクカーネル、マルチタスクモニターを実現する事ができる。　このマルチタスクカーネルは並行処理をサポートしたコンパイラのランタイムルーチンとして最低限必要な機能を実現したものであり、現在開発中のコンパイラにはプロセス間通信機構としてランデブーを採用する予定である。言語のモデルとしてはＡｄａを簡略化した物を考えている。ランデブーを実現する手法としてHabermann-Nassi法の変形を採用する予定である。Habermann-Nassi法はエントリを呼び出したタスクがそのエントリを実行するというアルゴリズムであるが、ここではエントリを実行するタスクはエントリを所有するタスクであるがスタックはエントリを呼び出したタスクのものを利用するというものである。こうすることにより引数をコピーする必要がなくなり、効率よく実現できるのではないかと思われる。

筑波大学におけるUnixをランデブーで記述する研究では引数をコピーするアルゴリズムを採っておりそのためタスクスイッチングのオーバヘッドが全CPUタイムの60%を消費するという。この手法でどの位改善できるかはわからないが少なくとも60%以下であることが期待される。