4º SBRC
RECIFE - 24 A 26 DE MARÇO 86

ARQUITETURA DE UM SUPORTE DE COMUNICACÃO BASEADO NA REDE LOCAL ETHERNET E NO UNIX

Joberto S. B. Martins
UFPB - Brasil

Gérard Noguez
Université Paris VI - France

RESUMO

Este artigo apresenta a concepção de um suporte de comunicação baseado na rede Ethernet e no Unix (V.7). O sistema desenvolvido permite a concepção de servidores e inclui um mecanismo de virtualizacão de recursos hardware. Uma arquitetura performante para o hardware é proposta. O software descrito é estruturado até o nível transporte. Assim sendo, nós consideramos a implementação de um handler Ethernet, do protocolo, dos mecanismos de sincronização e temporização e, finalmente, as estruturas de dados utilizadas são analisadas. Por último, o nível de transporte é introduzido através de uma apresentação resumida do mecanismo de "guichets" utilizado e da modelização das aplicações distribuidas.

1. HARDWARE

O hardware projetado consiste em uma interface rede suportando a norma Ethernet [MET-76] [XER-80] (fig. 1). A solução desenvolvida é baseada nos circuitos VLSI especializados 82586 (Coprocessador Ethernet) e 82501 (Interface Série Ethernet).

A solução proposta para a interface responde aos critérios essenciais de concepção de uma estação de trabalho ("workstation"), no sentido em que :

- a performance obtida é satisfatória (um "turnaround delay" de 4Mbits/seg. liquido é obtido para o nível 2 do protocolo),
- a compactação é garantida,
- a simplicidade de concepção contribui à redução dos custos de implantação.

A arquitetura implantada é baseada numa série de critérios inovadores :

- inicialmente, o coprocessador utiliza unicamente a memória partilhada ("common memory"), situada no módulo da unidade central de processamento,
- além do mais, a utilização de uma "capacidade de processamento" (microprocessador) na interface é considerada como sendo desnecessária. Isto, em função do fato que a solução VLSI utilizada é suficientemente "inteligente".

De uma maneira geral, a solução proposta é inovadora, em

parte, pela sua autonomia, e, também, pela elimininação dos níveis de armazenamento da informação ("buffering") entre a interface rede e a CPU. Este último aspecto a diferencia consideravelmente da maioria das soluções propostas para a arquitetura de interfaces rede [MIN-83] [FAN-84].

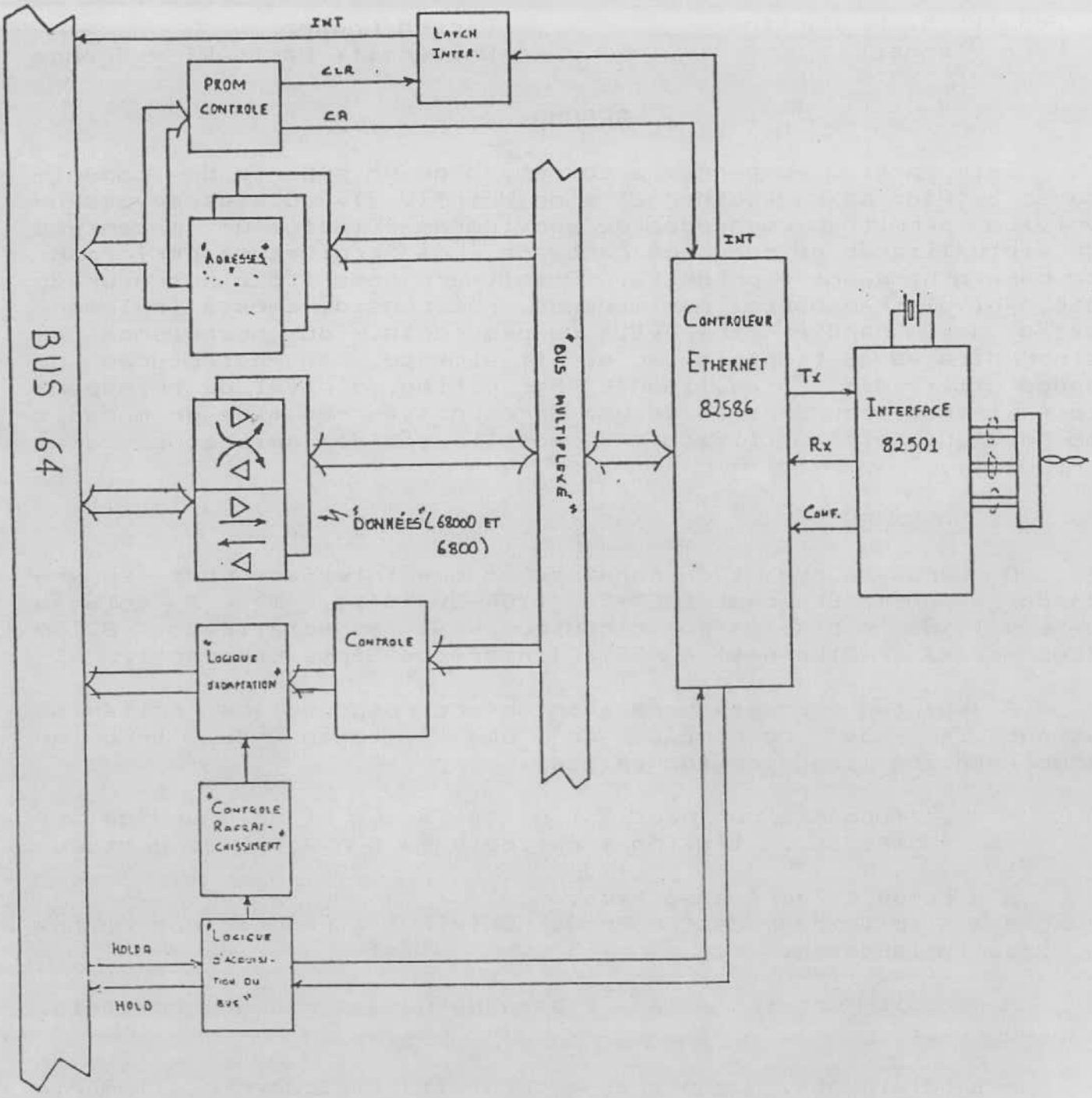

Fig.1: Arquitetura da Interface Ethernet

2. SOFTWARE

2.1 - OBJETIVOS E CLASSIFICAçÃO

De início, nós classificamos o sistema de comunicação desenvolvido como um sistema "pouco acoplado" ("loosely coupled") [PUJ-82] [HUT-82] que suporta a distribuição de recursos centralizados preservando a autonomia local [JOB-84].

Os objetivos principais aos quais nós nos propomos são os seguintes :

- primordialmente, a criação de um suporte de comunicação que permite a concepção de servidores (disco, impressora, correio eletrônico, ...),
- a implantação de um mecanismo de virtualização que permite a utilização eficaz dos recursos hardware (periféricos disco e fita) no contexto da rede,
- além do mais, este suporte deverá permitir a concepção de mecanismos de comunicação de alto-nível (protocolos).

As características principais que nós procuramos impor ao suporte de cominicação, através da opções de concepção tomadas são as seguintes :

- boa performance,
- simetria de concepção,
- facilidade de utilização do suporte desenvolvido (aspecto utilizador).

De uma maneira geral, as soluções tomadas consideram o contexto de aplicação no qual o suporte é inserido. Isto corresponde à considerar os seguintes fatores :

- o contexto rede local,
- os critérios de concepção de uma "estação de trabalho" ("workstation").

2.2 - ESTRUTURAçÃO

A estrutura do software que permite a concepção de aplicações distribuidas, considerando o contexto de rede local e estação de trabalho, é ilustrada na figura 2.

A solução adotada é baseda em níveis hierárquicos e constituida de três níveis principais :

- nível aplicação,
- nível transporte,
- nível de transmissão de pacotes.

O nível aplicação corresponde aos processos (servidores, aplicações, ...), utilizando um protocolo de aplicação definido pelo usuário.

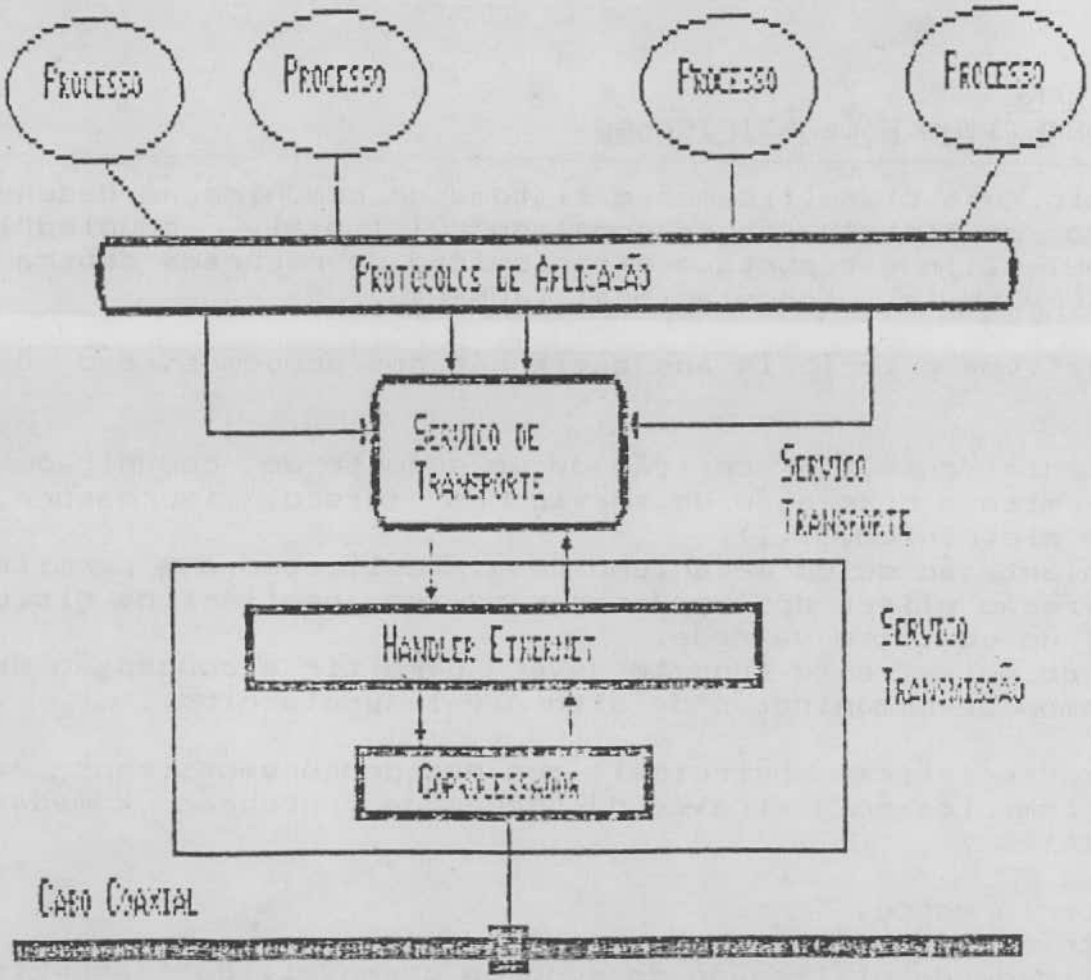

Fig. 2 : Modelizacão do Software

O nível transporte fornece o que nós chamamos de "serviço transporte" ou "serviço rede". Este serviço é implementatado pelas primitivas rede do sistema operacional, as quais permitem a comunicação entre processos distantes e não relacionados.

Enfim, o nível de transmissão de pacotes corresponde aproximadamente aos níveis 1 e 2 do modelo ISO (nível físico e enlace de dados). Na nossa implementação, ele é constituido pelo coprocessador Ethernet ("firmware") e por uma parte do "handler" Ethernet introduzido no sistema operacional.

É importante observar, que o nível rede (nível 3 do ISO) é considerado de funcionalidade nula na nossa implementação, em função da utilização de um protocolo de difusão de mensagens nas camadas inferiores do modelo (Ethernet).

Assim, a implementação do software segundo o modelo proposto, corresponde à concepção de um "handler" Ethernet e de um protocolo de transporte adaptado ao contexto das aplicações potenciais.

3. HANDLER ETHERNET

3.1 - CONSIDERAÇÕES GERAIS

A concepção do "handler" Ethernet considera as seguintes

opções técnicas :

- a interface Ethernet é vista pelo Unix como um periférico do tipo bloco ("bloc device") [THP-79] [RIT-79],
- o 'handler' utiliza, na medida do possível, as estruturas de dados normalmente utilizadas pelos periféricos tipo bloco (estrutura "buf", "cache"),
- nós utilizamos massivamente os mecanismos de base disponíveis no Unix (sincronização, temporização, ...).

De início, é importante observar que um periférico de rede é considerado normalmente como sendo do tipo caracter (não estruturado). Na realidade, nós optamos por uma estruturação do periférico Ethernet com o objetivo preciso de facilitar a concepção de um mecanismo de virtualização dos periféricos tipo bloco (disco, fita). Esta opção permite uma concepção simples e "quasi-transparente" do mecanismo citado [JOB-84], isto, em função do fato que nós utilizamos a estrutura "buf" como parte integrante do mecanismo de controle do 'handler' Ethernet.

As duas primeiras opções permitem uma polivalência do 'handler' em relação aos tipos de comandos que ele é capaz de tratar (fig. 3). Assim sendo, ele considera a execução dos comandos normais da rede (emissão e recepção) e, também, o mecanismo de virtualização dos periféricos bloco.

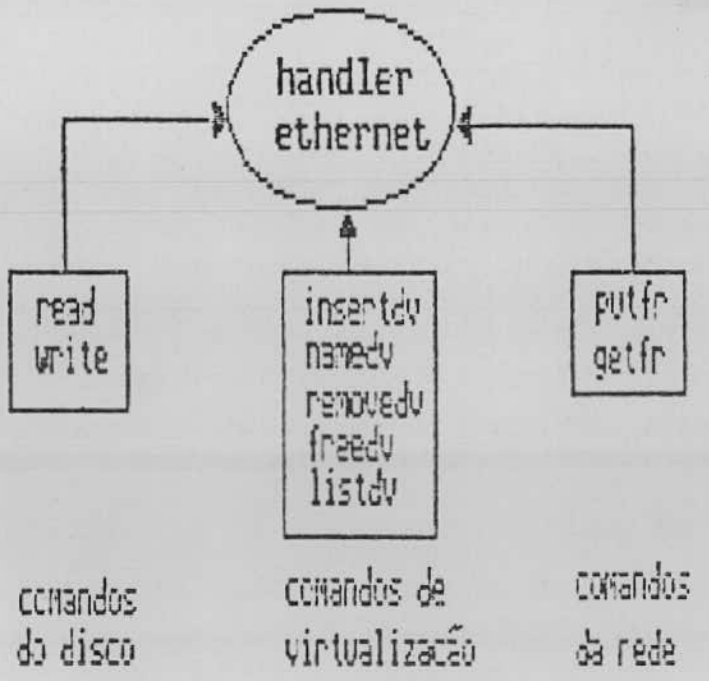

Fig 3. : Comandos do Handler

Esta é uma aproximação feita no sentido de fornecer ao usuário um máximo de serviços "off-the-shelf".

Uma outra questão à abordar, diz respeito aos serviços fornecidos pelo 'handler'. Nossa aproximação consistiu à utilizar uma filisofia oposta à argumentação "end-to-end" [SAL-81].

Assim, nossa solução comporta dois serviços ao nível do "handler" :

- serviço garantido,
- serviço não garantido.

O serviço garantido corresponde à execução pelo "handler" de um protocolo de comunicação que garante a transmissão de pacotes em sequência, sem erros ou duplicações. A introdução de um protocolo no espaço supervisor do sistema operacional é uma solução que propicia :

- uma flexibilidade para a realização de mecanismos (ex : circuito virtual) e protocolos (ex : transporte) de nível superior,
- uma performance de operação muito melhor que uma solução baseada na concepção do protocolo de comunicação no espaço usuário do Unix [MAR-84],
- a realização de um mecanismo de virtualização de periféricos.

Por outro lado, o serviço "não garantido" corresponde à um serviço do tipo datagrama, ou seja, um serviço onde a transmissão dos pacotes é feita sem garantia de sequênciamento, duplicações ou erros. Este serviço corresponde ao modelo encontrado na maioria dos "drivers" de rede.

4. TEMPORIZAÇÃO ("timeouts")

A introdução de um protocole de comunicação no "handler" implica necessariamente na utilização de um mecanismo de temporização. Este mecanismo é necessário à eliminação do caráter aleatório introduzido em recepção pelo suporte de transmissão utilizado e pela independência dos processos comunicantes.

A implementação das temporizações utiliza a função "timeout(func, arg, interval)", existente no núcleo do Unix.

Uma função especializada foi desenvolvida onde os fatores específicos do contexto rede são considerados :

- previsão do número de tentativas de reemissão,
- exclusão mútua de atividade de um sub-periférico da rede ("guichet"),
- indicação de inatividade do mecanismo.

Funcionalmente, este mecanismo é acionado sempre que uma solicitação de transmissão garantida de pacotes é posta ao "handler".

No que diz respeito ao resultado prático deste tipo de implantação, nós temos à salientar que :

- a solução funciona bem, sem penalisação acentuada da performance do sistema operacional, mesmo considerando um núme-

ro razoável de processos.
- entretanto, a utilização do mecanismo Unix, tal qual, implica em uma sub-utilização das estruturas "callout" [UNX-83] do sistema. En efeito, sempre que a emissão de um pacote é bem sucedida, a atividade do mecanismo de temporização correspondente sobrecarrega inutilmente o sistema operacional. Além do mais, a parametrização dos períodos de temporização ("timeouts") e a definição do tamanho da matriz "callout" é crítica. Isto, de maneira à evitar que o sistema fique bloqueado pelo fato de não ter encontrado nenhuma entrada disponível em "callout" ("clock.c").

Assim sendo, nós desenvolvemos uma função adicional, que permite a desativação do mecanismo de temporização sempre que o "handler" constata a execução completa do serviço solicitado.

5. SINCRONIZAÇÃO

A noção de sincronisação é diretamente associada à necessidade de cooperação existente entre os processos de um sistema. Em efeito, os processos necessitam de um mecanismo de exclusão mútua e de um mecanismo de transmissão de informação (ex : sincronização), de maneira à que eles possam cooperar.

No Unix, nós temos várias primitivas e mecanismos acessíveis direta ou indiretamente aos utilizadores (ex : primitivas "wait", "pipe", "signal", sincronização das E/S). Todos estes exemplos utilizam um mecanismo de base existente no núcleo, que corresponde às funçoes "sleep/wakeup".

Nossa solução consistiu à implementar o mecanismo de sincronização das primitivas da rede segundo o esquema ilustrado na figura 4 :

- o mecanismo de base é Unix ("sleep/wakeup"),
- os processos em emissão de mensagens utilizam o mecanismo de sincronização das E/S ("iowait/iodone"),

- os processos em recepção de mensagens são sincronizados pelas funções "ethwait/recepdone", especialmente desenvolvidas para a rede.

As razões que justificam a soluçao apresentada são as seguintes :

Em primeiro lugar, a emissão de mensagens utiliza os "buffers" livres do "cache" e a estrutura de controle "buf". Logo, é evidente que o par "iowait/iodone" possa ser utilizado sem grandes problemas. Além do mais, esta solução preserva a uniformidade do sistema em relação à concepção do mecanismo de virtualização e facilita consideravelmente a interface direta com os apelos supervisor "read/write" do sistema operacional.

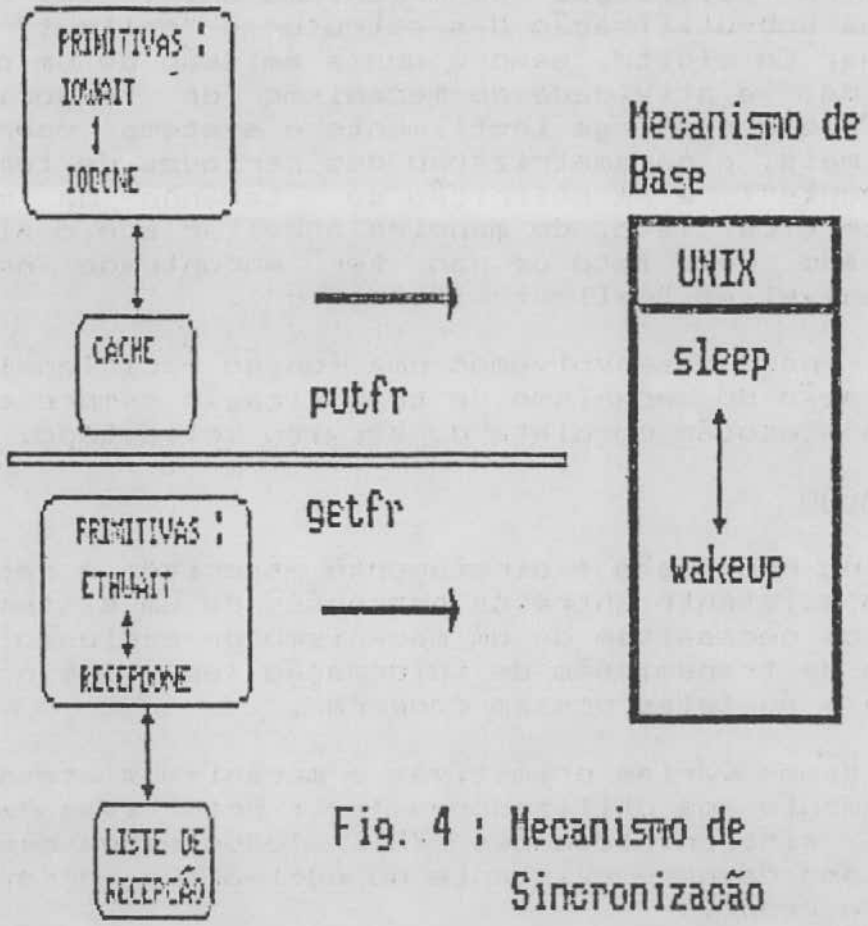

Fig. 4 : Mecanismo de Sincronização

No que diz respeito à recepção de mensagens, a implementação das rotinas especializadas se justifica pelo fato que :

- as primitivas "iodone/iowait" são inadequadas ao mecanismo de recepção,
- as estruturas de dados em recepção são diferentes das dos "cache".

As funções das rotinas criadas são as seguintes:

"ethwait" ---> faz um processo "dormir" ("sleep") num "guichet" durante a espera de um pacote,
"recepdone" --> "acorda" ("wakeup") o processo dormindo no guichet e passa o pacote recebido pela rede ao processo.

A figura 5 ilustra o princípio geral da operação descrita.

6. PROTOCOLO

O protocolo utilizado em nossa concepção leva em conta uma série de fatores intrínsecos do software desenvolvido :

- a contenção do processador central na execução das tarefas ("tasks") associadas aos protocolos dos diferentes níveis,

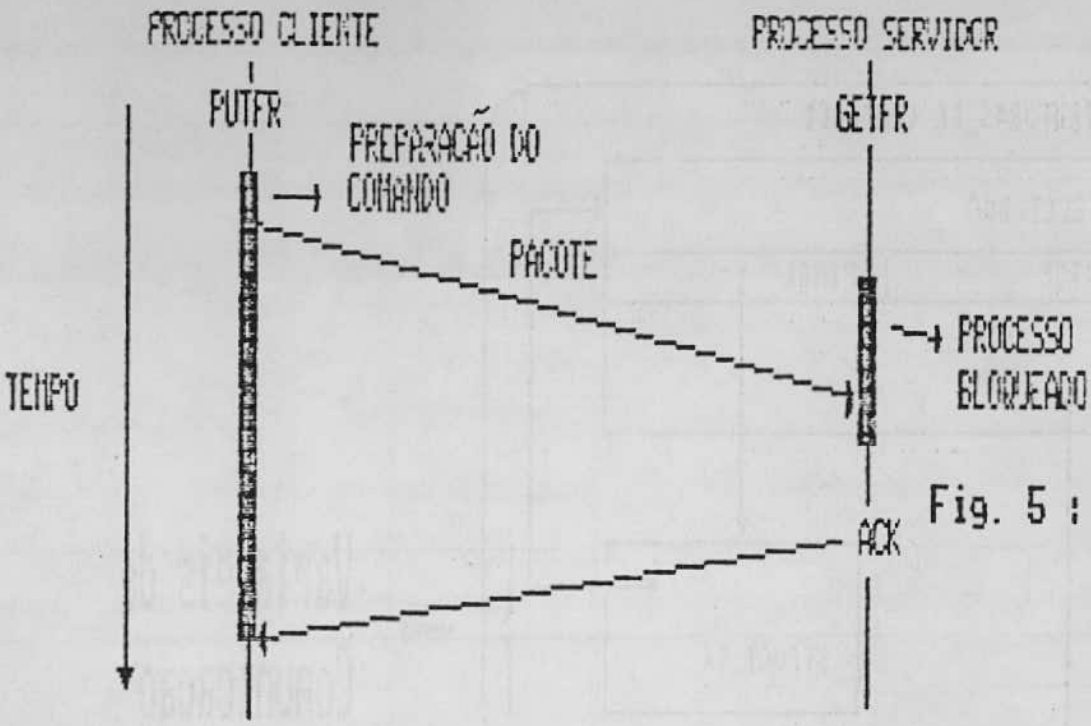

Fig. 5 : Sincronização dos processos

- a estrutura multi-níveis,
- o contexto multi-processo.

Assim sendo, nós optamos por um protocolo de janela móvel ("sliding window protocol") e fixamos o tamanho da janela em 1. A imposição deste tamanho de janela se justifica em parte pelas seguintes razões :

- a comunicação entre os processos é essencialmente sequencial,
- a utilização de uma janela maior implica na utilização de recursos (memória, CPU, ...) que, no contexto considerado, é limitante,
- Kritzinger [KRI-85], mostra no seu artigo que a classe dos protocolos "stop-and-wait" pode ser bastante eficiente sob o ponto de vista do usuário de um sistema de comunicação. No caso, a eficiência demonstrada nos pareceu satisfatória.

Finalmente, a utilização do mecanismo de "piggybacking" não foi considerada, em função da penalização adicional infligida pela implantação das temporizações. Além do mais, as interações resultantes do modelo cliente/servidor, introduzido pelo suporte de comunicação, são tais que a probabilidade de ter mensagens disponíveis para fazer o "piggybacking" seja pequena.

7. ESTRUTURAS DE DADOS

A operação da interface Ethernet é controlada pela estrutura de dados ilustrada na figura 6. A estrutura é constituida por :

- estrutura "buf",
- um "guichet",
- estrutura "req".

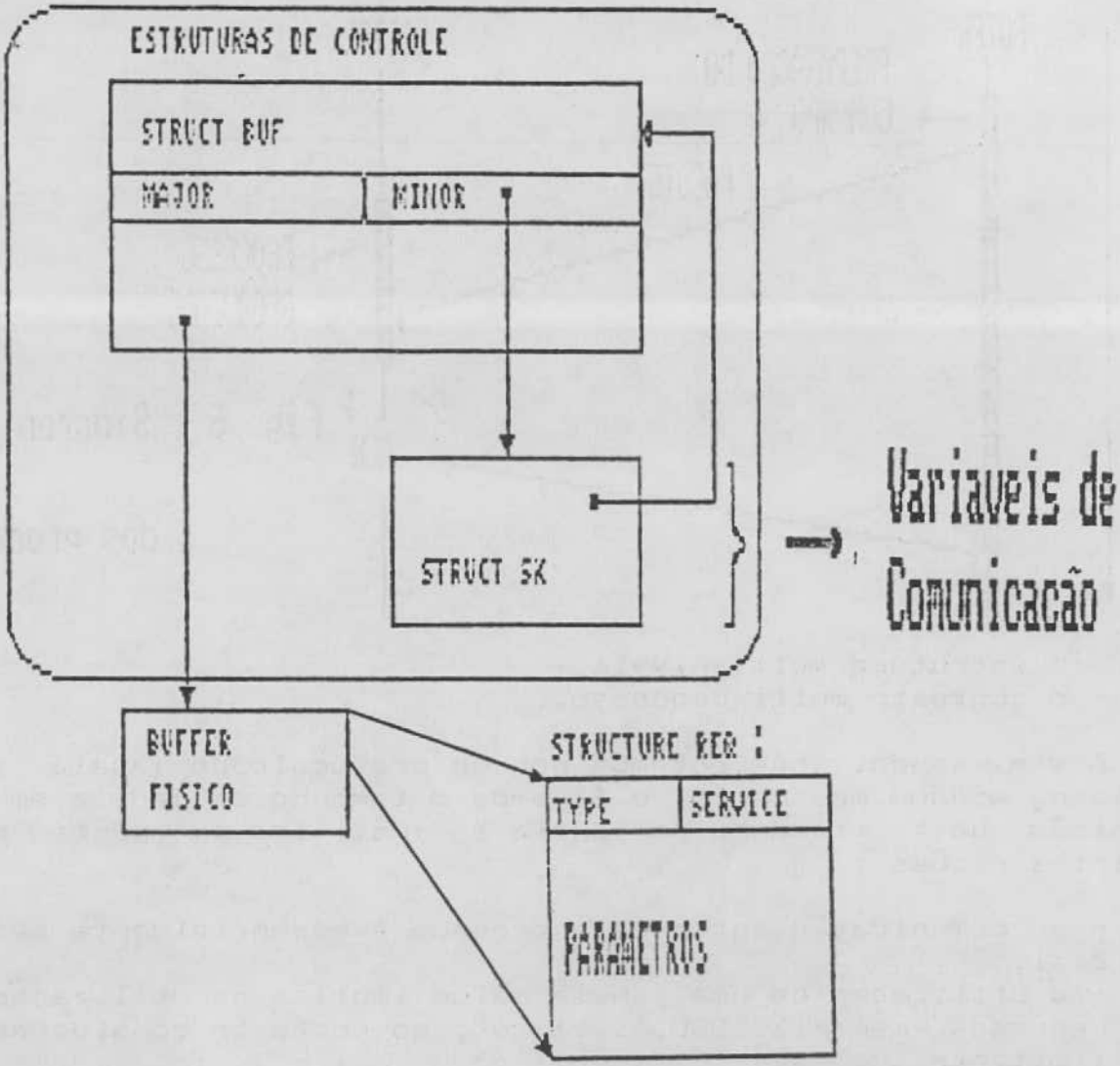

Fig. 6 : Estruturas de Controle do Handler

A estrutura "buf" faz a interface entre os processos solicitadores de serviço e o "handler" Ethernet instalado. As solicitações de E/S são colocadas ao "handler" numa fila FIFO (fig. 7) e o tipo da operação é definido em "buf". Na realidade, nós simplesmente definimos algumas novas operações em "buf", sem fazer nenhuma modificação na interpretação das outras variáveis. As novas operações definidas são as seguintes :

B_DATA ----> serviço garantido,
B_REPL ----> serviço não garantido,
B_CNTR ----> operação de controle da rede.

O "guichet" corresponde ao princípio dos "sockets" encontrado na concepção de outros sistemas distribuidos [LEF-85] [JOY-81]. Ele corresponde à un sub-periférico rede ("minor") e contém uma extensão da estrutura "buf" no sentido de permitir a comunicação entre processos distantes e não relacionados.

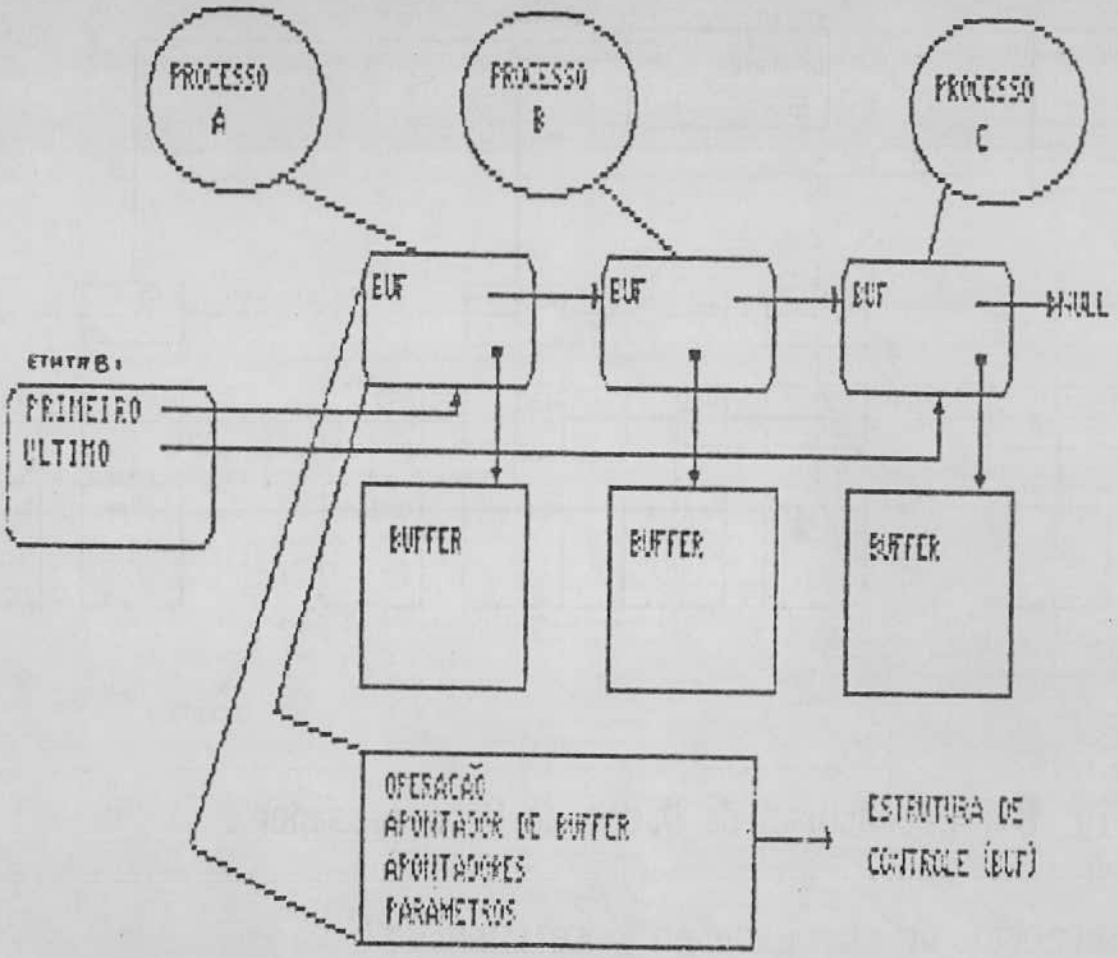

Fig. 7 : A Fila de espera (FIFO) do Handler

Enfin, a estrutura "req" permite a execução das operações de controle da rede (circuito virtual, comandos do coprocessador, diagnósticos, ...).

Por outro lado, as estruturas de dados utilizadas pelo coprocessador Ethernet (82586) são apresentadas na figura 8. De uma maneira geral, nós ressaltamos as seguintes opções utilizadas na implantação do modelo de gestão dos buffers de E/S :

- a estrutura de dados em emissão é uma "lista estática" [INT-83],
- a estrutura de dados em recepção é uma "lista dinâmica".

Em efeito, o caráter dinâmico da lista de recepção é fundamental para a obtenção de uma boa performance de operação do sistema de comunicação.

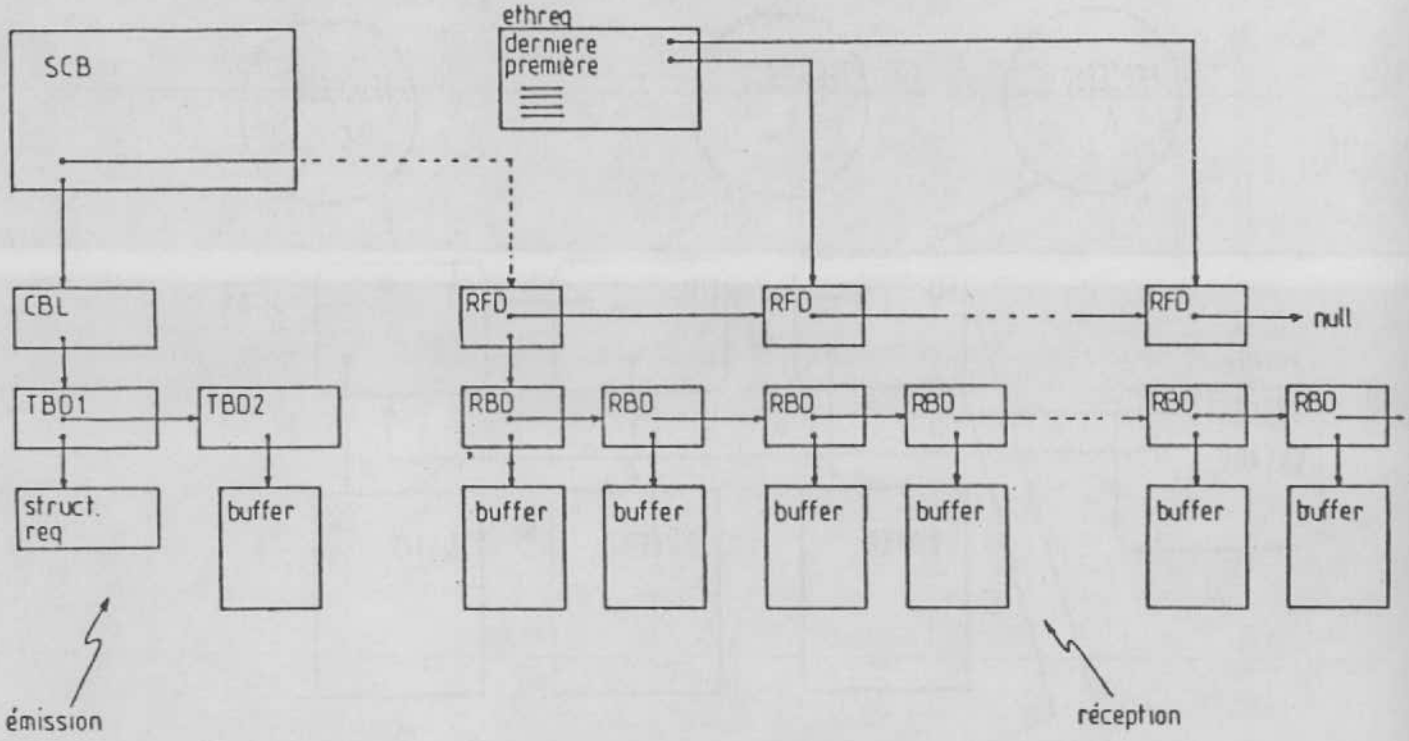

Fig. 8 : Estruturas de Dados do Coprocessador

8. O SUPORTE DE COMUNICAçÃO : INTRODUçÃO

O suporte de comunicação desenvolvido corresponde ao protocolo de transporte instalado no espaço supervisor do sistema operacional.

As seguintes considerações são válidas para este protocolo :

- ele não é compatível com protocolo definido pelo ISO [ISO-83a],
- nós consideramos na implantação um sub-conjunto dos serviços definidos pelo ISO para o protocolo de transporte classe 4 [ISO-83].

Em efeito, nós definimos um protocolo adaptado ao contexto das aplicações potenciais deste projeto (rede local, capacidade computacional das máquinas, homogeneidade, ...), sem considerar toda a generalidade da proposição do ISO.

Na nossa concepção, o nível 4 corresponde à interface rede visível aos processos utilizador e sistema. Nós consideramos que uma implantação até este nível é normalmente suficiente para a concepção de aplicações distribuidas [DAN-81] e, especificamente para as redes utilizando um protocolo do tipo CSMA-CD, um serviço classe 4 é recomendado [STO-83] [LAN-84].

Os serviços fornecidos pelo nível transporte são os seguintes :

- serviço circuito virtual,
- serviço datagrama.

Além do mais, as primitivas da rede (apelo ao supervisor) instaladas, são melhor adaptadas à utilização do modelo cliente/servidor na modelização das aplicações distribuidas.

O usuário dispõe do seguinte conjunto de primitivas para a concepção das aplicações distribuidas :

- "socket" -------> obtenção de um "guichet" da rede
- "vcsetup" ------> criação dos circuitos virtuais
- "putfr" --------> emissão de pacotes
- "getfr" --------> recepção de pacotes
- "lognet" -------> controle dos "servidores"

Igualmente, uma biblioteca rede é disponível de maneira à que o utilizador possa abstrair a parametrização dos apelos ao supervisor mencionados.

9. "GUICHET"

O acesso à rede é feito unicamente pelos "guichets". Eles são objetos protegidos acessíveis através das primitivas da rede. Os "guichets" são logicamente associados aos processos e são uma das componentes do endereço de transporte destes.

Em termos de estrutura de dados, o "guichet" armazena as informações necessárias à criação e manutenção da comunicação entre processos distantes. Ele contém :

- o protocolo utilizado,
- a descrição do processo servidor associado,
- semáforos de operação (status, temporização, sincronização, ...),
- estruturas de dados de emissão e recepção, ...

Os tipos de "guichets" disponíveis aos utilizadores são os seguintes :

- datagrama ---------> suporta um serviço do tipo datagrama,
- circuito virtual --> suporta um serviço circuito virtual,
- definido ----------> suporta "bootstrap" remoto.

Os "guichets" podem ser mapeados no sistema de arquivos do Unix, o que permite a utilização dos apelos ao supervisor "read/write" sobre estes.

10. CIRCUITOS VIRTUAIS

A criação dos circuitos virtuais com os processos servidores é realizada utilizando o apelo ao supervisor "vcsetup" ou as

funções da biblioteca rede indicadas abaixo :

```
#include        <net.h>
vcconn (servidor, guichet)      /* criação da conexão */
char *servidor;                 /* nome do servidor */
int guichet;                    /* número do guichet local */

vcdcon (servidor, guichet)      /* liberação da conexão */
```

As seguintes características gerais são válidas :

- a representação dos recursos da rede (servidores) é feita utilizando "nomes" ("by name"),
- a conexão com o processo servidor é feita utilizando um mecanismo de difusão de mensagens ("broadcasting"). Isto permite a "migração" dos processos servidores entre as máquinas da rede,
- nós assumimos a existência de um processo administrador de conexão aos circuitos virtuais nas máquinas tendo servidores instalados,
- a representação dos "nomes" dos recursos é glogal, sob a forma de um arquivo de configuração do tipo "/etc/passwd" (arquivo de configuração dos usuários Unix).

11. MODELIZAÇÃO DAS APLICAÇÕES

Na nossa concepção, os recursos da rede são representados por um conjunto de regras de cooperação definidas no contexto de um modelo cliente/servidor.

De uma maneira resumida, o modelo pode ser visto da seguinte maneira :

- os processos servidores esperam indefinidamente nos seus "guichets" a chegada das solicitações de serviço.
- os processos clientes solicitam os serviços. A solicitação é feita à dois níveis. Inicialmente, o processo cliente se conecta ao processo servidor através de um circuito virtual. Em seguida, o protocolo de aplicação é executado utilizando as primitivas da rede para troca de mensagens de dados ("putfr/getfr").

Neste ponto, é importante observar que o protocolo de aplicação é definido pelo utilizador.

12. SERVIDORES

A representação e o controle dos servidores nas máquinas da rede é realizada por um conjunto de funções da biblioteca derivadas do apelo ao supervisor "lognet".

Um resumo das diferentes possibilidades implantadas segue :

- auto-ativação de um servidor,
- ativação de um processo servidor à partir de um processo

administrador,
- conexão dos servidores aos circuitos virtuais à partir de um processo administrador,
- verificação do "status" dos servidores (ativação, "guichet" utilizado, número do processo).

13. ILUSTRAÇÃO DO SUPORTE DE COMUNICAÇÃO DESENVOLVIDO

Abaixo, nós mostramos através de um pequeno exemplo em linguagem C, a simplicidade de programação das aplicações distribuidas utilizando o suporte de comunicação desenvolvido :

```
/*
 * exemplo simplificado de um programa usuário
 * se conectando à um servidor "SERV".
 */

#include <net.h>

main()
{
     int  sk, n, cnt;
     char *cbuf;

          .
          .
     sk = skvcget();                  /* obtenção guichet circuito
          .                              virtual */
          .
     n = vcconn("SERV", sk);          /* conexão ao servidor */
     if( n != -1) {                   /* execução do protocolo de
               .                         aplicação */
               .
          n = putvc(sk, cbuf, cnt);      /* emissão mensagem */
               .
               .
          n = getvc(sk, cbuf, cnt);      /* recepção mensagem */
               .
               .
     }
     n = vcdconn("SERV", sk);         /* liberação da conexão */
          .
          .
}
```

14. CONCLUSÃO

Neste projeto, nós tentamos combinar duas características que, no nosso modo de ver, são importantes para a concepção de uma "estação de trabalho" numa rede local : boa performance e simplicidade de implementação.

No que diz respeito ao hardware, a solução proposta permite uma banalização da utilização da rede. Além do mais, ela assegura

um nível de performance satisfatório no contexto das aplicações potenciais.

O suporte de comunicação proposto é uma solução que procura estabelecer uma melhor performance pela eliminação da superposição de camadas de software na implementação do software. Por outro lado, um mecanismo de virtualização de recursos hardware é também suportado pela concepção [JOB-84].

O conjunto das soluções propostas permite fundamentalmente uma evolução da versao V.7 do Unix no sentido de suportar a concepção da aplicações distribuidas.

REFERÊNCIAS BIBLIOGRÁFICAS

[DAN-81] André A. S. Danthine, "Design Principles of Communications Protocols," Proceedings of the IFIP (TC-6) / CSI Conference on Networks, India, 1981.

[FAN-84] Alain Fanet et al., "Vers une Machine d'Interconnexion de Réseau," Congrès des Nouvelles Architectures pour les Communications, Paris, 1984, pp. 103-110.

[HUT-82] D. Hutchison and P. Corcoran, "Building Local Area Networks to Ethernet Specification," Computer Communications, vol. 5, nº 1, february, 1982, pp. 12-16.

[INT-83] Intel, "LAN Component Users Manual," 1983.

[ISO-83] International Organisation for Standardization, "Transport Service Definition," Draft International Standard ISO/DIS 8072 - ISO/TC 97, 1983.

[ISO-83a] International Organisation for Standadization, "Connection Oriented Transport Protocol Specification," Draft International Standard ISO/DIS 8073 - ISO/TC 97, 1983.

[JOB-83] Joberto Martins, "Un Réseau Local Ethernet sous Unix pour l'Application en CAO," Rapport de Recherche, Univ. Paris VI, octobre, 1983.

[JOB-84] Joberto Martins et Gérard Noguez, "Virtualisation des Ressources Matérielles à l'Aide d'un Réseau Local Ethernet sous Unix," Congrès des Nouvelles Architectures pour les Communications, Paris, 1984, pp. 227-233.

[JOY-81] William Joy and Roberty Fabry, "An Architecture for Interprocess Communication in Unix," Departement of Electrical Engineering and Computer Science, University of California Berkeley, Draft of June 22, 1981.

[KRI-85] P. S. Kritzinger, "Analysis of ARQ Protocol's in a Multiprocess Environment," Research Report - IBM Zurich Research Laboratory, Switzerland, 1985.

[LAN-84] Sylvain Langlois et Jean-Pierre Ansart, "Implémentation du Protocole de Transport ISO Class 4 sur un Réseau Local du type CSMA-CD, "Congrès des Nouvelles Architectures pour les Communications, Paris, 1984.

[LEF-85] Samuel J. Leffler et al., "A 4.2 BSD Interprocess Communication Primer," Departement of Electrical Engineering and Computer Science, University of California, Berkeley, Draft of March 22, 1985.

[MAR-84] Bernard Martin et al., "Problèmes Posés par le Dévéloppement de Logiciels Réseau sous Unix," Congrès des Nouvelles Architectures pour les Communications, Paris, 1984.

[MET-76] Robert M. Metacalfe and David R. Boggs, "Ethernet : Distributed Packet Switching for Local Computer Networks," Communications of ACM, vol. 19; pp. 395 - 403, 1976.

[MIN-83] Michael Minnich and Charles J. Coton, "An Evaluation of Two Unibus Controllers," 8th. Conference on Local Computer Networks, Minnesota, 1983, pp. 29-36.

[PUJ-82] Guy Pujolle, "La Télématique : Réseaux et Applications," Eyrolles, Paris, 1982.

[RIT-79] D. M. Ritchie, "The Unix I/O System," Unix User's Manual, Bell Laboratories, vol. 2, 1979.

[SAL-81] J. H. Saltzer et al., "End-to-End Arguments in System Design," Proceedings of the 2º Int. Conf. on Distributed Computer Systems, april, 1981.

[STO-83] Daniel P. Stokesberry, "Use of ISO Class 4 Transport on Local Area Networks," 1983.

[THP-79] K. Thompson, "Unix Implementation," Unix User's Manual, Bell Laboratories, vol. 2, 1979.

[UNX-83] Fonte Unix, versao 7, Bell. Labs.

[XER-80] Xerox, DEC, Intel, "The Ethernet : A Local Area Network, Data Lynk Layer and Physical Layer Specifications," version 1.0, 1980.