Implémentation de protocoles de communication ISO sur un mini-6 sous UNIX

Bernard Feydel

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

CENTRE RÉGIONAL ASSOCIÉ DE

GRENOBLE

MÉMOIRE

présenté en vue d'obtenir

le DIPLÔME D'INGÉNIEUR C.N.A.M.

en

INFORMATIQUE

par

Bernard FEYDEL

Implémentation de protocoles de communication ISO sur un mini-6 sous UNIX

Soutenu le 23-10-84

JURY

PRESIDENT :

MEMBRES :

Implémentation de protocoles de communication ISO sur un mini 6 sous UNIX

Bernard Feydel

Mémoire de thèse C.N.A.M.

SOMMAIRE

Ce mémoire a été réalisé dans le cadre du projet ESPRIT (Europeen Strategic Program for Research and development in Information Technologies). L'objectif général est de mettre en place un réseau d'échange d'informations entre les centres de recherche qui coopèrent au sein de la Communauté Européenne avec une ouverture vers les grands réseaux extérieurs en particulier américains. Ce réseau a comme support un environnement UNIX (1) et il doit respecter les normes définies par l'ISO (2) en matière d'interconnexion de systèmes ouverts.

Dans les chapitres 4 et 5 de ce mémoire nous détaillerons la partie implémentation du Protocole HDLC (3) d'une part dans un coupleur programmable et d'autre part dans le noyau du système UNIX sur un mini 6.

(1) UNIX est une marque déposée de Bell Laboratories
(2) Inernational Standards Organization
(3) High level Data Link Control procedure

1. Présentation du projet.

1.1. Introduction

Ce chapitre nous permettra de situer le travail effectué pour ce mémoire dans son environnement, c'est à dire le projet européen ESPRIT.

1.2. Le projet ESPRIT.

Ce projet est né de la volonté des états membres de la Communauté Européenne de relever le défi lancé par les USA et le Japon pour la maitrise des nouvelles technologies de l'information pour la prochaine décennie en créant un espace européen de la recherche. Ses objectifs consistent à créer en Europe un potentiel de ressources en recherche et développement grâce à la mise en place d'un programme international financé à hauteur de 750 millions d'Ecus, soit 5,25 milliards de francs, par le budget communautaire plus un montant équivalant pris en charge par les participants au projet.

Ce programme prévoit de faire collaborer des équipes de recherche sur des projets complémentaires par rapport aux différents programmes nationnaux et pour une période de dix ans

- La microélectronique
- La technologie du logiciel
- L'intelligence artificielle
- L'environement bureautique
- La production intégrée par ordinateur

En plus de ces cinq thèmes, le projet d'infrastructure EIES (European Information Exchange System) doit mettre en place un réseau d'échange d'informations entre les chercheurs.

Pour piloter le projet ESPRIT, une commission de consultation définit et met à jour des plans de travail qui seront présentés chaque années devant un "concile" pour approbation. Les projets ont été classés en deux grandes catégories :

[a] Les projets de type A pour des actions de grande envergure en recherche et développement. Ces projets nécessitent une infrastructure importante et il est prévu de leur allouer 75% des ressources totales.

[b] Les projets de type B doivent privilégier la recherche individuelle par rapport à l'approche système et ils s'appuient sur une infrastructure plus flexible.

Le projet ESPRIT a débuté en 1982 par une étude préliminaire qui a permis de dégager les grands thèmes de recherche et de leur affecter les ressources correspondantes. Un premier plan de travail recouvrant une période de cinq ans propose une évaluation que nous reproduisons dans le tableau 1.1 ci-dessous.

thèmes proposés	années 1	2	3	4	5	total
microélectronique	186	258	360	410	256	1.670
logiciel	177	317	343	318	285	1.440
intelligence artificielle	140	281	392	441	441	1.695
systèmes bureautique	210	310	440	390	100	1.450
C.A.O.	121	216	215	220	172	944

tableau 1.1: répartition des ressources (années/homme)

L'étude préliminaire a également mis en évidence le besoin de fournir aux différents partenaires du projet des outils leur permettant d'échanger très facilement toute sorte d'informations. La diversité des systèmes d'information utilisés par les intervenants a conduit les responsables à définir le projet pilote EIES.

1.3. Le projet pilote E.I.E.S.

1.3.1. Objectifs du projet

Ce projet a pour mission de développer en trois ans la technologie nécessaire à la mise en place d'un réseau d'échange d'informations entre les états membres. L'essentiel du travail doit porter sur la fourniture de véritables services aux intervenants du programme ESPRIT mais certains aspects pourront rester dans le domaine de la recherche comme par exemple l'exploration des moyens de communication à très hautes performances.

Pour répondre aux recommandations énoncées, le système EIES doit pouvoir offrir plusieurs types de connections logiques avec un environnement différent du modèle "OSI" comme le montre la figure 1.2 :

- Une ouverture vers les systèmes non "EIES" avec une possibilité de stockage intermédiaire de l'information.
- Pour pouvoir accèder à certains réseaux ne répondant pas aux normes "ISO" et utilisés en recherche et développement, il faut prévoir la construction de "passerelles".
- D'autres "passerelles" sont également prévues pour communiquer avec des réseaux privés utilisant des architectures différentes suivant les besoins des utlisateurs ESPRIT.
- Accès aux services télématiques non "OSI" des administrations des PTT.

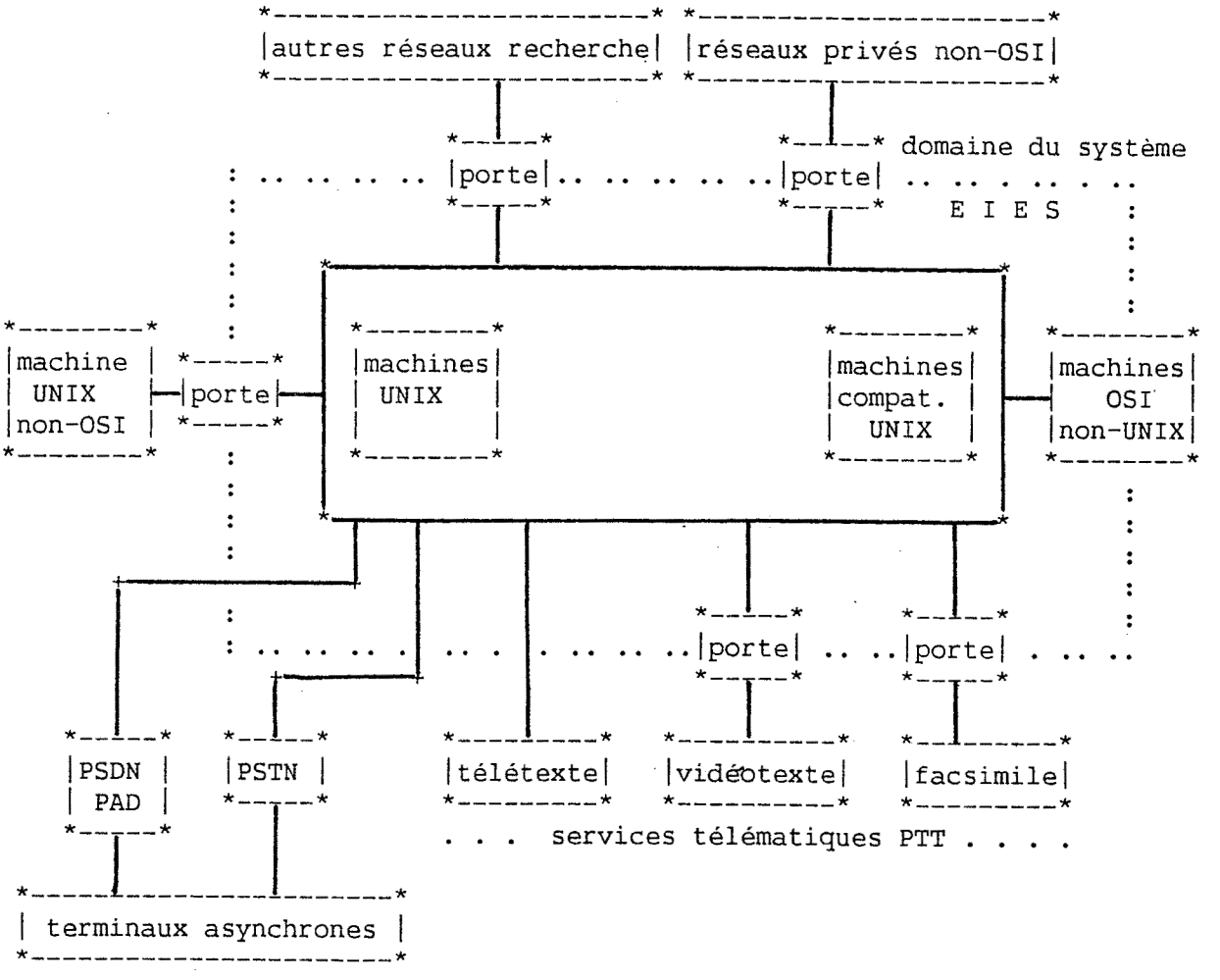


figure 1.2: Connexions logiques dans le système EIES

1.3.2. Les choix dans EIES:

L'étude préliminaire réalisée en 1983 a permis de faire le point sur l'état de l'art en matière de système d'exploitation et de techniques de communication afin de pouvoir choisir les supports du réseau.

1.3.2.1. Le système d'exploitation:

Compte tenu des contraintes liées aux objectifs fixés au départ, il est vite apparu qu'il fallait construire le système EIES en s'appuyant sur un système d'exploitaion commun. Parmi ces contraintes nous pouvons citer :

- Le développement de logiciel distribué
- L'interface utilisateur commun
- Faciliter la portabilité du logiciel. (en relation avec le développement d'un environnement d'outils communs portables dans le projet pilote)
- La possibilité d'interfacer d'autres environnements sur un même site.

Parmi tous les systèmes d'exploitation étudiés, le choix s'est porté sur UNIX au regard des avantages qu'il présente comme :

- Une conception orientée vers la fabrication de logiciel et la préparation de document.
- Sa disponibilité sur un grand nombre de machines
- Son indépendance vis à vis des constructeurs d'ordinateurs
- sa portabilité
- Son utilisation très répandue dans les milieux de la recherche

1.3.2.2. L'architecture du système de communication.

Le système de communication tel qu'il a été défini doit résoudre de nombreux problèmes de compatibilité compte-tenu de son ouverture et de l'hétérogénéité des matériels connectés. Il faut par exemple gérer différents niveaux d'implémentation des standards en fonction des PTT des pays concernés, résoudre l'incompatibilité entre les processeurs dont le mode d'adressage est basé sur le mot et enfin compenser le manque de produits capables de combiner du texte et de l'image.

Pour toutes ces raisons il est apparu nécessaire de s'appuyer au maximum sur les normes internationnales définies en matière d'architecture de système de communication. C'est donc le modèle proposé par l'ISO qui a été retenu (norme ISO 98) et le système EIES devra intégrer les protocoles OSI (Open System Interconnection) dans UNIX.

1.3.3. Planning prévisionnel du projet.

Comme nous l'avons vu ci-dessus, le projet pilote EIES doit se dérouler sur une période de trois ans en plus de l'étude préliminaire. Une des recommandations importantes demande aux contractants de fournir un réseau opérationnel à chacune des étapes. Nous allons examiner ces différentes phases.

1.3.3.1. L'étude préliminaire:

La fin de l'étude préliminaire a démontré la possibilité de faire coopérer les équipes de développement des entreprises concernées avec la mise en service d'un premier réseau utilisant le réseau téléphonique commuté publique qui permet d'offrir aux utilisateurs deux types de service:

- la connection à distance sur une machine hôte en utilisant la commande UNIX "cu" (Call Unit).
- Le transfert de fichier entre deux sites distants avec la mise en œuvre de l'utilitaire UNIX uucp (unix to unix copy).

1.3.3.2. Année 1 du projet.

L'objectif fixé pour la première année (1984) est de mettre à disposition des participants du projet ESPRIT un système d'échange d'information plus complet, ce qui suppose:

- le support des protocoles "X25" et "ETHERNET" pour la couche réseau du modèle ISO.
- la mise en place de la couche "TRANSPORT". (classe 3 au dessus d'X25 et classe 4 au dessus d'ethernet).

- l'intégration de la couche "SESSION".
- la disponibilité d'outils standards comme le courrier électronique ou le transfert de fichiers.

1.3.3.3. Les années 2 et 3 du projet.

Pour ces deux années, il est demandé aux contractants de faire progresser les services offerts sur le système jusqu'au niveau défini dans le cahier des charges du projet EIES et d'ouvrir les possibilités d'accès aux autres réseaux, systèmes et terminaux extérieurs, en intégrant la totalité des protocoles du modèle OSI.

Au terme de cette étape, le système devra être:

- ouvert
- performant
- d'une grande fiabilité

Enfin cette dernière étape prévoit également des recherches dans le domaine des liaisons à gros débit ou des liaisons par satellite.

1.4. La répartition des tâches.

La direction du projet est assurée par la compagnie Bull laquelle prend également en charge le secrétariat. Pour la construction du système, chacun des participants développera, sur la machine qu'il utilise, le logiciel nécessaire au support des protocoles X25 et ethernet (ceux ci étant très dépendant de la machine).

La couche transport du système sera fabriquée par la société ICL (Angleterre) et la couche session sera fourni par Bull. La société Olivetti (Italie) doit élaborer un système de gestion de l'adressage tandis que GEC (Angleterre) sera chargée d'adapter la commande de transfert de fichier (uucp) au dessus de la couche transport.

Notons que la société SIEMENS (Allemagne) et l'INRIA (Institut National de Recherche en Informatique et en Automatique) participent également au projet EIES, l'INRIA étant plus spécialement chargé d'adapter le système d'exploitation SOL sur du matériel SM90 comme alternative possible à UNIX.

Un des principes de fonctionnement consiste à ce que chaque équipe participant au projet adapte sur son propre site le logiciel développé par les autres équipes.

2. La normalisation et le modèle de référence de l'ISO.

2.1. La normalisation dans les réseaux.

2.1.1. Généralités.

Ces dernières années, les réseaux informatiques ont connus un développement très important pour différentes raisons. Cet essor est lié aux besoins accrus en matière de communication du fait de la volonté de décentralisation qui se dessine. D'autre part il est rendu possible par l'évolution des techniques et des matériels disponibles sur le marché. Face à cet accroissement rapide, l'utilisateur se trouve confronté à de nouveaux problèmes :

- d'une part assurer la cohérence de ses installations (possibilité de communication d'un système à un autre)
- d'autre part préserver sa liberté de choix face aux constructeurs de matériels informatiques.

La meilleure réponse à ce type de problèmes consiste à mettre en place toute une série de "règles du jeu" que devront respecter les parties prenantes, c'est ce qui a été réalisé avec la normalisation. Le problème est similaire à celui qui se pose aujourd'hui pour les magnétostopes, chaque constructeur développant ses propres standards, l'utilisateur est contraint d'acheter tel ou tel type de cassette vidéo.

L'urgence d'une normalisation a encore été accélérée par les progrès importants effectués dans le domaine de la technologie avec l'avènement des microprocesseurs, l'apparition de terminaux "intelligents" ou l'intégration de protocoles complets dans de simples composants électroniques.

La définition des normes en matière de réseaux n'implique pas seulement les constructeurs de matériel informatique et les utilisteurs mais il faut considérer que les PTT sont également partie prenante puisqu'ils ont un monopole des télécommunications dans la plupart des pays. Le marché de l'informatique dépassant largement les frontières, il était indispensable que les normes soient internationales.

2.2. Les différents organismes de normalisation.

Dans cette section nous citerons rapidement les principaux organismes de normalisation CORN[81] et le partage des responsabilités.

le CCITT: (Comité Consultatif International pour le Télégraphe et le Téléphone) il est constitué des administrations des PTT ou d'organisations privées reconnues. Le CCITT étudie plus particulièrement la normalisation concernant les voies de transmission, les équipements terminaux de circuit de données et les procédures pour établir, maintenir et libérer une liaison dans un réseau commuté.

L'ISO: ISO:'u-10m .br (Organisation Internationale de Normalisation) c'est une sorte de fédération des organismes nationaux de normalisation. Le partage des responsabilités entre l'ISO et le CCITT est fixé par l'avis A20 du CCITT.

l'ECMA: ECMA:'u-10m .br (European Computer Manufacturers Association) c'est le troisième organisme représentatif sur le plan international et il regroupe les grands constructeurs d'équipements informatiques en Europe.

Les principaux utilisateurs ont également formés des associations pour faire des propositions de normes.

La définition de normes s'effectue en plusieurs étapes par exemple dans le cas de l'ISO le cheminement est le suivant:

- Les organisations nationales comme l'AFNOR, le BSI ou l'ANSI présentent des propositions au niveau d'un sous-comité de l'ISO
- A la suite d'un vote favorable, l'avant-projet devient un projet de norme
- Le projet de norme est ensuite soumis au Comité technique qui vote à son tour
- si le projet est adopté il est enfin soumis au Conseil de l'ISO pour devenir une norme internationale de l'ISO.

Si la gestation des normes peut paraître longue, il ne faut pas oublier que cela représente un travail très complexe du fait de l'évolution rapide des techniques et de l'imbrication entre le traitement et la transmission de l'information.

2.3. Le modèle de référence de l'ISO.

(pour l'interconnexion de systèmes ouverts)

2.3.1. Préambule.

La description d'un modèle théorique nécessite une définition précise des termes utilisés et pour ce faire, nous renvoyons le lecteur au document AFNOR[82].

L'objectif de ce modèle est de fournir une base commune pour l'élaboration de normes sur l'interconnexion des systèmes. Il permet de situer chacune d'elle par rapport à un ensemble cohérent. Un système sera considéré comme "ouvert" lorsqu'il est capable de communiquer et de coopérer avec d'autres systèmes par le simple fait qu'il accepte les normes appropriées. Cela n'implique nullement une technologie ni des moyens de connexion particuliers.

Le modèle de référence concerne les échanges d'information entre systèmes ouverts et non pas le fonctionnement interne de ces systèmes. La figure 2.1 nous montre comment les transferts d'information entre les systèmes sont fait par l'intermédiaire de supports physiques d'interconnexion.

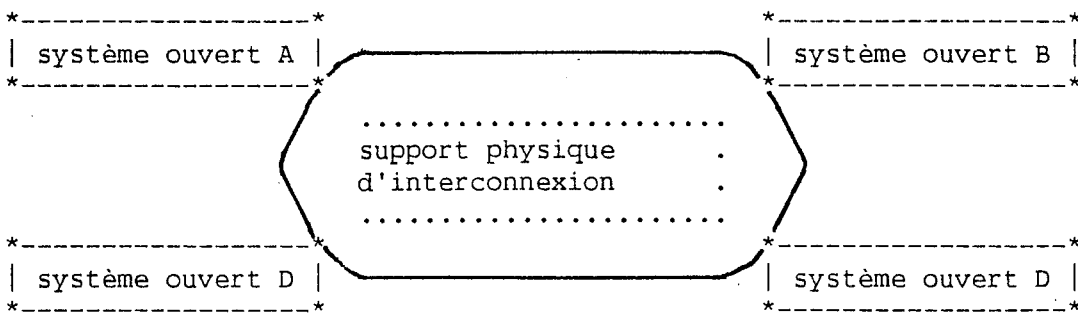


Figure 2.1: Connexion par des supports physiques.

La modèlisation s'est effectuée en deux étapes :

- La définition des éléments de base des systèmes ouverts abstraits et des grandes lignes de l'organisation et du fonctionnement. Cette étape conduit au modèle de référence OSI.

- La formulation d'une description précise du fonctionnement qui aboutit aux normes "services et protocoles d'interconnexion de systèmes ouverts".

2.3.2. Description du modèle.

2.3.2.1. Architecture.

L'architecture de base du modèle de référence est la structuration en couches. Cette technique part du principe que chaque système est logiquement composé d'un ensemble ordonné de sous-systèmes. L'ensemble des sous-systèmes de même rang noté (N) constitue la couche (N). La figure 2.2 nous montre la représentation d'un tel modèle.

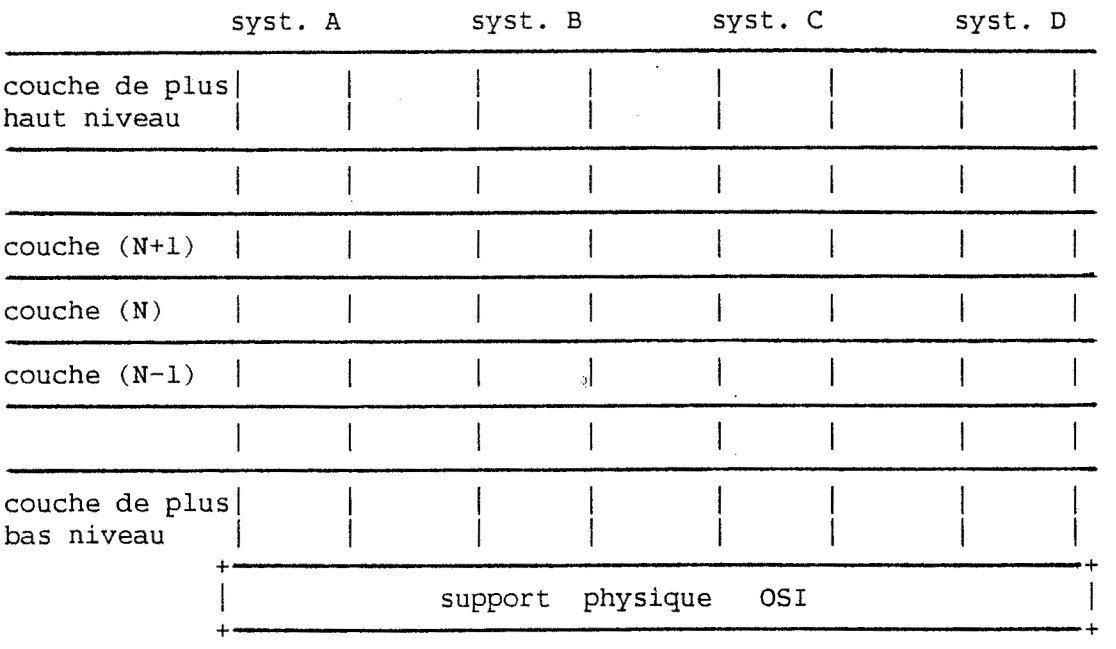


Figure 2.2: Structuration en couches.

Nous considérons que chaque couche (N) fournit des services (N) aux entités (N+1) de la couche (N+1). Chaque service peut être adapté en choisissant une ou plusieurs "facilités". Les entités (N) d'une couche peuvent coopérer en communiquant au moyen des services fournis par la couche (N-1). La coopération entre entités (N) est régie par un ou plusieurs protocoles (N) comme l'indique la figure 2.3.

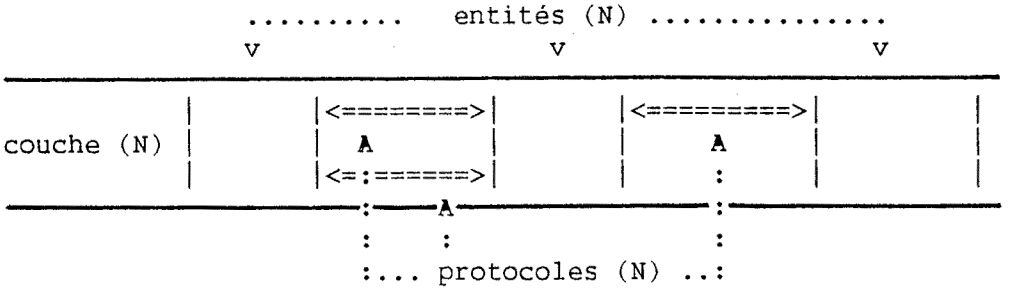


Figure 2.3: Protocoles.

2.3.2.2. Communication et Identificateurs.

Des entités (N+1) peuvent communiquer par l'intermédiaire d'une association dans la couche (N) en suivant un protocole (N). Cette association est appelée "connexion" (N) entre au moins deux "points d'accès" à des services (N).

Une "adresse (N)" identifie le point d'accès à des services (N) particuliers auxquels une entité (N+1) est liée. Nous utiliserons une "fonction de routage" pour traduire l'adresse (N) d'une entité (N+1) en un chemin permettant d'atteindre cette entité.

2.3.2.3. Fonctionnement d'une couche.

Nous avons vu que la communication entre des entités d'une même couche passe par une connexion, il faut donc étudier les caractéristiques de ces connexions:

- Etablissement et libération d'une connexion.

  Les entités (N) échangent des éléments nécessaires du protocole, si la connexion (N-1) n'est pas disponible, les entités homologues (N-1) doivent en établir une. Ce raisonnement se poursuit vers les niveaux inférieurs jusqu'à la rencontre soit d'une connexion disponible, soit du support physique d'interconnexion.

- Multiplexage et éclatement.

  Dans une couche, les connexions (N) sont mises en correspondance avec des connexions (N-1) selon trois possibilités:

  . une-à-une

  . plusieurs-à-une; on parle dans ce cas de multiplexage

  . une-à-plusieurs; il s'agit alors d'un éclatement

- Transfert de données.

  Le transfert de données peut mettre en œuvre plusieurs fonctions telles que:

. échange de données normales ou exprès

. contrôle de flux

. segmentation, groupage et concaténation

. maintien en séquence.

- Traitement des erreurs.

  Nous trouvons trois fonctions essentielles dans le traitement des erreurs:

  . Une fonction d'accusé de réception

  . Une fonction de détection et de notification d'erreurs

  . Une fonction de réinitialisation

2.3.3. Les différentes couches retenues dans le modèle.

L'ISO a retenu sept couches pour représenter son modèle de référence :

[1] la couche Physique,

[2] la couche Liaison de données,

[3] la couche Réseau,

[4] la couche Transport,

[5] la couche Session,

[6] la couche Présentation,

[7] la couche Application.

Les couches numérotées de un à six de même que le support physique contribuent à l'élaboration progressive des services de communication. La couche sept ou application contient les entités d'application qui coopèrent dans l'environnement OSI.

Certains systèmes ne remplissent qu'une seule fonction, celle de faire suivre les données qu'ils recoivent vers un autre système, on parle dans ce cas de "systèmes relais". Nous trouvons un tel exemple dans la figure 2.4.

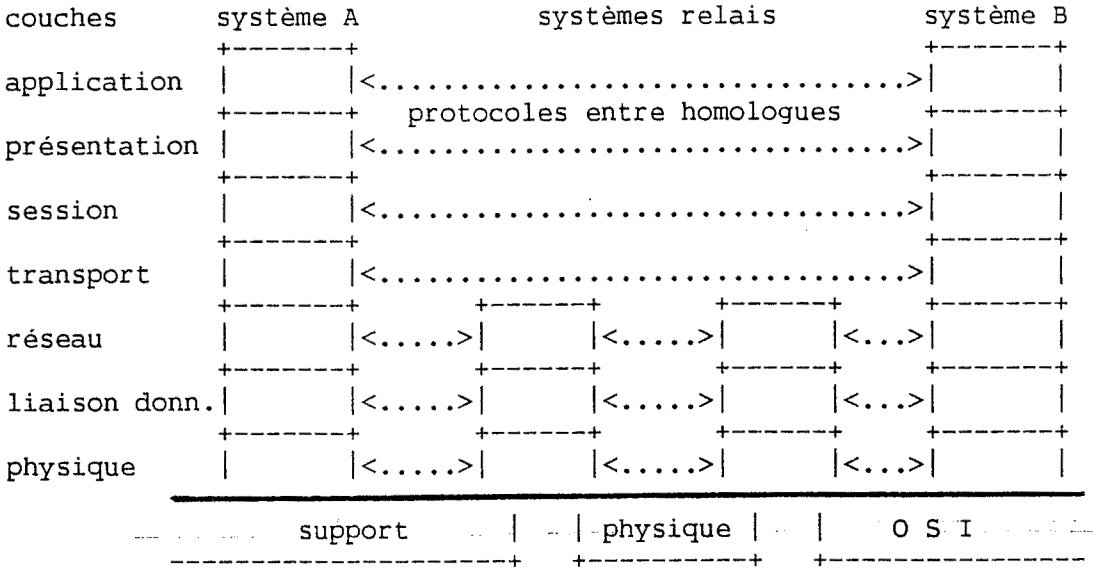


figure 2.4: Modèle de référence avec systèmes relais.

2.3.3.1. La couche physique:

Elle fournit les moyens nécessaires à l'activation, au maintien et à la désactivation des connexions physiques destinées à la transmission de bits entre entités de liaison de données. Cette couche interface directement avec le support physique OSI, ce qui apparait dans la figure 2.5. C'est la couche physique qui réalise éventuellement une commutation de circuit.

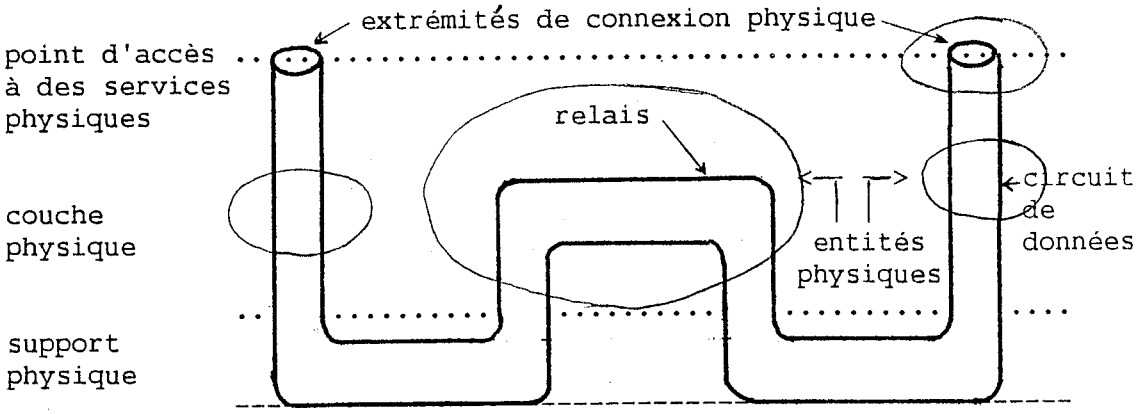


Figure 2.5: Interconnexion de circuits de données.

2.3.3.2. La couche liaison de données.

Cette couche a pour mission d'assurer l'acheminement sans erreur de blocs d'information sur des liaisons de données. Elle est capable de détecter et de corriger dans certains cas les erreurs pouvant se produire dans la couche liaison physique. C'est à ce niveau que se situe la procédure HDLC que nous allons décrire ultérieurement.

2.3.3.3. La couche Réseau.

Son rôle est de garantir le transfert transparent de données entre des entités de transport. Elle assure à la couche transport une indépendance vis à vis des moyens de communication utilisés dans les couches inférieures. Le service fourni à chacune des extrémités d'une connexion de réseau est identique. La qualité du service offert est négocié à l'établissement d'une connexion de réseau.

La couche réseau remplit entre autres les fonctions suivantes:

- les fonctions de routage
- les fonctions de relais
- le multiplexage des connexions de réseau

- la segmentation et le groupage des données.

Le niveau 3 de la norme X25 (accès aux circuits virtuels) est l'une des composantes de la couche réseau.

2.3.3.4. La couche Transport.

Cette couche a pour but d'optimiser l'utilisation des services de réseau disponibles. Elle assure un transfert de données transparent entre entités de session avec la responsabilité du contrôle de "bout-en-bout". La couche transport concerne les systèmes extrémité de l'interconnexion des systèmes ouverts (OSI). Elle est composée de plusieurs classes qui correspondent à des niveux de service différents. La classe est choisie à la connexion de transport.

2.3.3.5. La couche Session.

C'est à ce niveau que nous trouvons la responsabilité de la mise en place et du contrôle du dialogue entre des tâches distantes. La couche session offre des services comme:

- l'établissement de la connexion avec négociation des paramètres
- la rupture normale ou impérative (possibilité de perte de messages)
- échange de données normales ou exprès
- synchronisation avec "marquage" des points de reprise
- resynchronisation de processus sur incident
- segmentation des messages

2.3.3.6. La couche Présentation.

Elle regroupe des fonctions d'intéret général en rapport avec la représentaion et la manipulation de données structurées à l'intention des programmes d'application. C'est la couche présentation qui contient par exemple le protocole d'appareil

virtuel (PAV).

2.3.3.7. La couche Application.

Elle donne aux processus "utilisateurs" les moyens d'accèder à l'environnement OSI. Etant la seule couche du modèle à fournir directement des services aux processus d'application, elle offre nécessairement tous les services OSI.

2.3.4. Remarques.

L'aspect pédagogique du modèle de référence peut être considéré comme le plus important. En effet, il permet par ses définitions précises et son architecture en couches de décomposer un problème très complexe, voir inextricable, en une succession de sous-problèmes beaucoup plus faciles à appréhender.

Par contre, une implémentation complète de ce modèle représente une quantité de code très importante (overhead) qui ne sera peut-être pas justifiée dans tous les cas. La présentation très modulaire permettra néanmoins de s'adapter aux différentes situations par exemple en ignorant simplement certaines couches. Il restera à trouver un équilibre entre les performances du système, sa fiabilité et le niveau de service rendu.

3. L'environnement UNIX et mini-6

3.1. Le système d'exploitation UNIX.

3.1.1. Généralités.

C'est vers 1970 que Ken Thompson des Bell Laboratories a commencé à développer le système UNIX qui s'est voulu dès le départ: BOUR[82], KAAR[83]

- multi-utilisateurs et interactif,
- portable et flexible,
- avec un système de gestion de fichier hiérarchisé,
- intégrant des outils puissants (langage de commandes, formatage de documents, etc ...).

En 1977 Dennis Ritchie entreprit de réécrire UNIX en langage "c", langage de haut niveau qui donnera à ce système sa grande portabilité et sa facilité de maintenance, deux de ses atouts importants aujourd'hui.

A l'origine, UNIX a été développé sur un matériel Dec PDP 7 et sa première transposition s'effectua en 1976 sur un matériel Interdata 8/32. Depuis, UNIX a été porté sur de nombreuses machines, des plus petites comme les microprocesseurs (Zilog Z80 Z8000, Motorola MC68000 ou Intel 8086) jusqu'aux plus grosses unités centrales (IBM 370 ou Amdahl 470). La commercialisation d'UNIX a débuté en 1975 avec la distribution des premières licences par Western Electric. UNIX a connu le succès en premier lieu dans les universités et les milieux de la recherche, les industriels restant prudents à son encontre. Puis progressivement, à partir des années 1980, le monde des affaires a réalisé qu'UNIX pouvait apporter une aide précieuse pour résoudre une grande variété de problèmes. Aujourd'hui le succès de ce système est incontestable et son avenir offre de belles perspectives.

Malgré toutes ses qualités, UNIX n'est pas universel et ce système présente quelques lacunes, en particulier ce n'est pas un système "temps réel". Cela signifie simplement qu'il est mal adapté aux applications qui nécessitent des temps de réponse très

brefs à des évènements extérieurs. UNIX ne possède pas non plus un système de communication inter processus IPC (Inter process communication) très développé (cet aspect sera précisé dans le chapitre 6).

3.1.2. Le cœur du système ou noyau (Kernel).

Une des idées intéressante d'UNIX est d'avoir "sorti" du système toutes les fonctions qui n'y sont pas indispensable pour les réaliser dans des modules extérieurs appelés fonctions standards. De ce fait, le noyau d'UNIX qui regroupe tous les modules résidents du système, reste d'une taille limitée mais en contrepartie il n'offre que peu de services. D'une manière plus générale, UNIX ne fait rien directement pour l'utilisateur, ce dernier devant exécuter des utilitaires intermédiaires.

Dans sa version 7, le noyau d'UNIX comporte environ 10.000 lignes de code écrit en langage "C" et 1000 lignes codées en assembleur, ce rapport pouvant varier suivant le type de machine utilisé, mais il reste qu'un tel volume de code peut facilement être compris et maintenu par une seule personne. Cela représente un avantage considérable par rapport à d'autres systèmes d'exploitation.

Le travail effectué par le noyau se divise en deux grandes parties:

- La gestion des processus
- La gestion des périphériques

La première partie qui comprend le "scheduler" est chargée de répartir les ressources de l'unité centrale entre les processus actifs. Cette fonction est essentiellement réalisée à partir de tables (table des processus, zone utilisateur par processus) et de l'horloge du système qui délimite les tranches d'exécution. Le code qui assure cette tâche est suffisamment standard pour être portable sur différents matériels.

A l'opposé, la deuxième partie du noyau UNIX est très dépendante de l'environnement puisqu'elle doit être capable de gérer des périphériques très variés (console système, unités de disque, lignes de communication, etc...). Dans ce domaine UNIX met en œuvre un mécanisme original qui permet d'exécuter un module de gestion des entrée-sortie spécifique en fonction du type de périphérique concerné. Ce mécanisme connu sous le nom de

"driver" appelle les différents programmes d'après le contenu d'une table de configuration. Ce système présente un double avantage:

- D'une part il permet de gérer tous le protocole lié à l'utilisation d'un périphérique donné dans un module indépendant des autres.

- D'autre part il rend le système adaptable à tout environnement par simple mise à jour de la table de configuration et éventuellement écriture d'un nouveau "Driver".

Il existe actuellement peu de systèmes où il est possible de réaliser aussi facilement qu'avec UNIX la prise en compte d'un nouveau périphérique, qu'il soit physique ou logique (pseudo périphérique). Nous verrons plus tard que ce mécanisme a été largement utilisé pour la réalisation de ce projet.

3.1.3. Le système de gestion de fichiers.

Le concepteur d'UNIX a cherché à présenter une gestion de fichiers qui soit pratique d'emploi. La réponse à ce critère a été de construire un système hiérarchisé qui autorise un regroupement logique des fichiers par centres d'intérêt à l'intérieur de répertoires ou "Directories".

Dans un système multi-utilisateurs, il est très important de pouvoir contrôler les droits d'accès aux données (informations ou programmes). A ce niveau UNIX permet de gérer les droits d'accès pour tous les types de fichiers avec une sélection possible entre lecture, écriture ou exécution.

Il existe un nouveau type de fichier dans UNIX appelé fichiers spéciaux et qui permet d'associer des droits d'accès et un "Driver" à chaque périphérique de la configuration en place. Par contre UNIX n'offre pas plusieurs méthodes d'accès aux fichiers.

En résumé, nous trouvons trois catégories de fichiers dans UNIX:

- Les fichiers ordinaires qui contiennent soit du texte (suite de caractères ASCII), soit du binaire (par exemple modules exécutables).

- Les répertoires ou "directories" qui sont des listes de

fichiers regroupés logiquement.

- Les fichiers spéciaux qui cataloguent les périphériques de la configuration.

Pour terminer sur les fichiers, il faut savoir que les données sont stockées sur les supports physiques par blocs de 512 octets, ces blocs n'étant pas obligatoirement contigus. Le système peut localiser l'information grâce à une table appelée "inode" qui contient les adresses des blocs d'information en plus d'autres données comme la longueur du fichier, son mode d'accès ou son propriétaire.

Ce principe permet une extension rapide des fichiers, sans remise en cause de l'information déja enregistrée. Par contre il introduit un surplus de traitement pour la gestion des tables d'inodes.

3.2. L'environnement logiciel dans UNIX

Un des attraits d'UNIX est de pouvoir offrir à ses utilisateurs toute une panoplie d'outils logiciels qui constituent un véritable environnement de fabrication de logiciel. Nous ne pouvons bien entendu pas tous les citer, mais nous mentionnerons ceux qui nous semble être les plus intéressants. La figure 3.1 nous montre la structure de cet environnement.

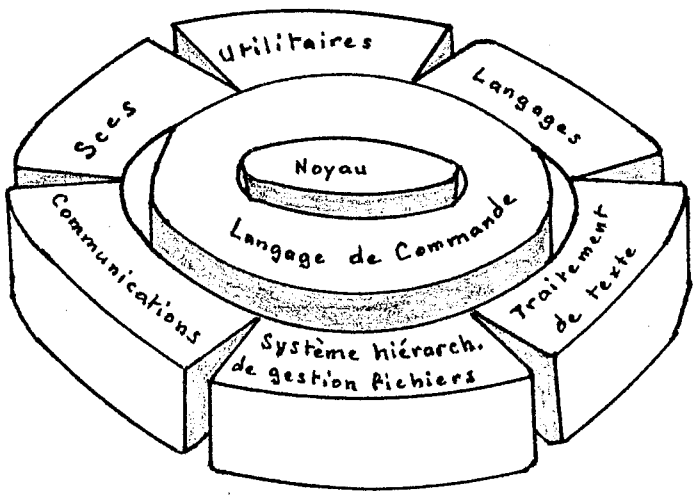


Figure 3.1 : l'environnement UNIX.

3.2.1. Le langage de commandes (Shell).

Dans UNIX, l'interprêteur de commande n'est pas inclus dans le noyau comme c'est souvent le cas dans d'autres systèmes, mais développé à l'extérieur. Le "Shell" est à la fois un interprêteur interactif de commande et un véritable langage de programmation du niveau commande.

Il comprend :

- des structures de contrôle (if, for, while, case)
- des variables utilisateurs ou automatiques
- des possibilités étendues de substitution de chaines de caractères et de passage de paramètres

Le Shell offre également des mécanismes intéressants comme la redirection des entrées-sorties ou les tubes ou "pipes" (canal unidirectionnel avec synchronisation entre deux processus).

Avec toute ces fonctionalités, le langage de commande permet d'automatiser au maximum le plus grand nombre d'applications.

3.2.2. Les outils de développement.

Dans cette catégorie UNIX contient un éventail remarquable d'outils originaux qui permettront à un concepteur de formaliser ses connaissances sur le produit qu'il développe.

Un programme comme 'make' peut aider l'utilisateur à résoudre le problème de coordination et de synchronisation des différents modules d'une application. Il accepte des spécifications qui définissent les relations existant entre les modules ainsi que les actions à entreprendre pour assurer leur mise à jour.

Avec 'SCCS' nous trouvons une gamme de programmes systèmes qui permet de maintenir et de documenter des programmes utilisateurs tout en assurant une liaison entre les différentes versions cataloguées d'un logiciel.

Citons enfin deux outils très puissants offerts par UNIX 'Lex' et 'Yacc' qui sont des générateurs d'analyseurs lexicographique et syntaxique. Les programmes générés par "Lex" ont pour but de découper un flot d'entrée en motifs (Token) fournis par l'utilisateur. Ces motifs sont associés à des expressions régulières grâce à un fichier de spécifications.

"Yacc" permet de construire des modules capables de vérifier la structure d'un flot d'entrée déja découpé en motifs. Les règles décrivant cette structure sont spécifiées dans une grammaire. Il est possible d'associer des actions à chacune des règles sous forme de sous-programmes.

La combinaison de ces deux outils avec des programmes utilisateurs autorise la construction de systèmes complexes tels que les compilateurs.

3.2.3. Le traitement de texte.

Le traitement de texte est l'une des fonctions les plus importantes dans UNIX, si bien qu'elle a été intégrée dans le système dès sa conception. La production de documents se fait à plusieurs niveaux par l'intermédiaire d'outils modulaires :

- Le premier niveau consiste à manipuler facilement les fichiers de texte, ce qui est facilité par l'utilisation du langage de commande très souple et des éditeurs de texte (mode ligne ou écran) disponibles sous UNIX.

- La phase suivante doit permettre à l'utilisateur de présenter des documents selon ses désirs. Cette fonction est réalisée par plusieurs utilitaires comme :

  - "Eqn" qui facilite la mise en page des formules mathématiques

  - "Tbl" qui est utilisé dans le formatage des tableaux

  - "Nroff/Troff" qui est le principal outil de traitement de texte. Il accepte des directives qui sont insérées dans le texte et qui automatisent la mise en page avec l'espacement horizontal et vertical, l'indentation, l'hyphénation, la justification, le numérotatage et bien d'autres fonctions.

Comme illustration des possibilités du traitement de texte dans UNIX, nous pouvons citer ce mémoire qui a été édité par "Nroff" sur imprimante laser DEC.

3.3. Le Mini-6.

3.3.1. Son architecture.

Le mini-6 CII[80] est construit autour d'un simple bus asynchrone à cycle interruptible appelé "Mégabus" sur lequel sont connectés des unités indépendantes, ce qui rend son architecture très modulaire. Parmi les principales unités connectables nous trouvons:

- Le processeur central
- Le contrôleur de mémoire
- les différents contrôleurs de périphériques
- des processeurs de communication multilignes
- etc ...

La figure 3.2 représente une configuration possible du mini-6.

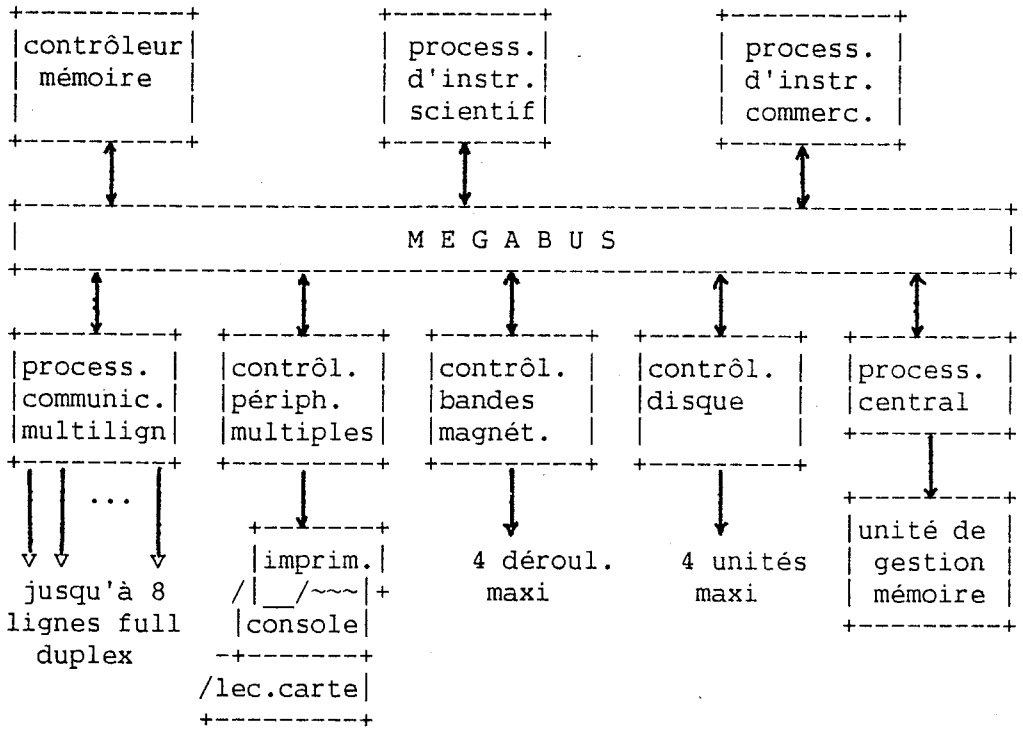


Figure 3.2 : configuration d'un mini-6 modèle 43.

La communication entre les unités est réalisée selon le cycle DMA (Direct Memory Access). Dans ce cas , le processeur central initialise le transfert d'information dans un programme canal qui peut ensuite s'exécuter de manière indépendante.

3.3.2. La gestion des interruptions.

Dans le mini-6, il existe 64 niveaux d'interruption qui sont générés soit par l'horloge temps réel, soit par les unités périphériques, par exemple à la fin d'un transfert. Suivant un niveau de priorité défini par programme, les interruptions sont prises en compte par le processeur qui exécute un programme d'interruption avant de revenir à son activité initiale. Le contexte est sauvegardé dans une zone mémoire particulière l'ISA (Interrupt Save Area).

Les déroutements sont assimilables à des interruptions logicielles car ils utilisent un mécanisme sensiblement identique appelé "Traps". Les appels système dans UNIX (system call) sont

implémentés sur le mini-6 par l'intermédiaire des "Traps".

3.3.3. La gestion mémoire.

Le mini-6 peut être équipé en option d'une unité de gestion mémoire (mmu) qui réalise la segmentation de la mémoire et assure une protection d'accès à trois niveaux : lecture, écriture et exécution. Il s'agit dans ce cas d'une protection physique alors qu'UNIX se charge de la protection logique.

la figure 3.3 montre comment l'espace adressable peut atteindre un million de mots.

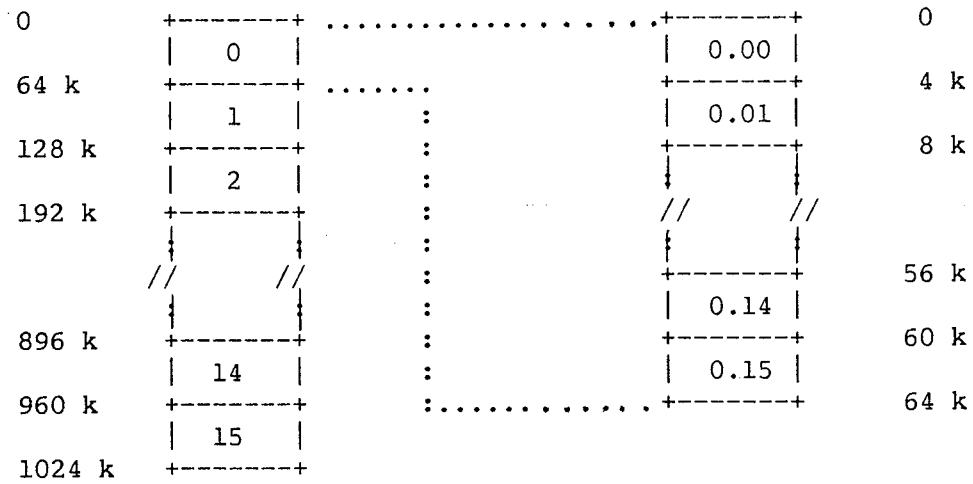


Figure 3.3 : Segmentation de la mémoire par la mmu.

4. La procédure HDLC.

4.1. Préambule.

Comme nous l'avons vu au cours du chapitre 1, il a été décidé de raccorder les différents sites du système de communication EIES par l'intermédiaire des réseaux publiques de transmission de données offrant un service de commutation par paquets. Pour ce faire, nous devons respecter l' avis X25 du CCITT X25[84] qui définit les interfaces et protocoles à mettre en œuvre sur les trois niveaux inférieurs du modèle de référence ISO (physique, liaison de données et réseau) comme nous le montre la figure 4.1.

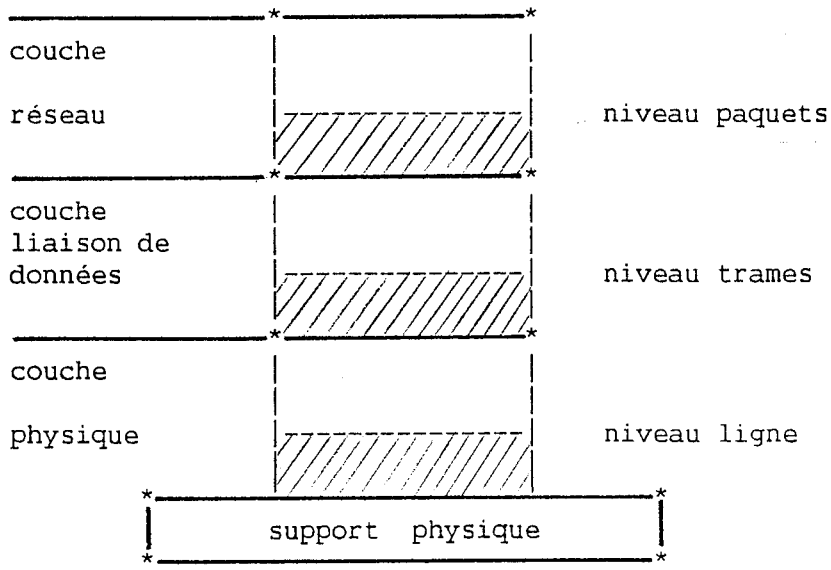


Figure 4.1 : domaine d'application de l'avis X25

La mise en œuvre de cette norme sur le mini-6 a représenté une part importante de la participation du centre de recherche de la BULL à Grenoble dans le cadre du projet EIES. Ce chapitre ainsi que le suivant s'attacheront à décrire plus particulièrement le développement du niveau liaison de données.

4.2. Définition

Les procédures d'accès à la liaison de données pour l'échange d'information sont regroupées sous le nom d' HDLC (de l'anglais: High Level Data Link Control). Elles s'appliquent à une communication point à point avec transmission bidirectionnelle simultanée synchrone. La procédure HDLC comporte des fonctions de détection d'erreur et de régulation de flux.

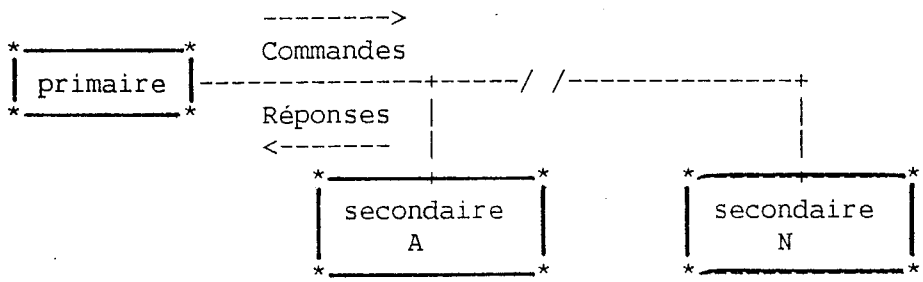
Dans le mode de transmission "synchrone", la reconnaissance des caractères se fait au niveau des blocs ou "trames". La procédure HDLC est dite basée sur l'élément binaire par opposition aux procédures basées sur le caractère qui nécessitent la réservation de certains codes pour la commande de la liaison (par exemple dans le mode de base). De ce fait la procédure HDLC est indépendante de l'alphabet utilisé au cours de la liaison.

La procédure HDLC comporte deux classes de base (équilibrée et non équilibrée) qui permettent de définir trois configurations logiques comme le montre la figure 4.2. WEISS[83] :

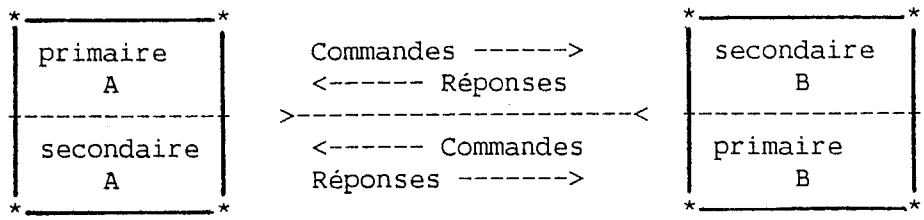
- Avec la classe "équilibrée" la liaison de données est établie entre deux stations combinées qui utilisent le mode ABM (Asynchronous Balanced Mode) dans une configuration point-à-point. Les contrôles sont identiques aux deux extrémités de la liaison et chaque station est responsable de la rœupération des erreurs du niveau liaison de données.

- Dans la classe "non équilibrée" la liaison de données se réalise entre une seule station primaire et une ou plusieurs stations secondaires. Il existe deux modes opératoires possibles : NRM (Normal Response Mode) ou ARM (Asynchronous Response Mode). Dans cette classe, seule la station primaire est responsable de la récupération des erreurs du niveau liaison de données.

Liaison "NON-BALANCEE" (UN)



Liaison "SYMETRIQUE" (UA)



Liaison "BALANCEE" (BA)

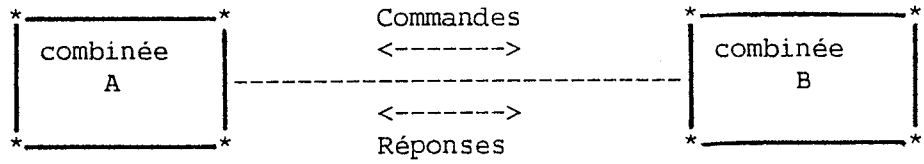


Figure 4.2: Configuration de liaisons logiques HDLC.

Remarques :

Le réseau TRANSPAC met en œuvre la configuration de type "équilibrée" pour la liaison de données avec la procédure LAPB.

Les trois classes de procédure ainsi définies (UN), (UA) et (BA) contiennent un répertoire de commandes et de réponses de base plus des extentions fonctionnelles qui représentent des options.

La normalisation de la procédure HDLC comporte trois parties :

- la structure de trame,

- les éléments du protocole,
- les mises en œuvre particulières.

Dans les paragraphes qui suivent nous utiliserons l'abréviation ETTD pour désigner l'équipement terminal de transmission de données (station connectée au réseau).

4.3. La structure des trames HDLC.

Dans la procédure HDLC, l'information est véhiculée à l'intérieur de trames qui sont considérées comme une suite d'éléments binaires. La figure 4.3 nous montre la structure des trames.

Fanion	Adresse	Commande	Inform.	F.C.S.	Fanion
01111110	8 élém.	8 élém.	*	16 élém.	01111110

(*) nombre indéterminé d'éléments binaires

figure 4.3: format d'une trame HDLC.

4.3.1. Le Fanion.

Toutes les trames doivent commencer et se terminer par un fanion (séquence de délimitation de trame). Notons qu'un même fanion peut fermer une trame et en ouvrir une autre. Le fanion sert à la synchronisation des trames.

4.3.2. Le champ d'adresse.

Le champ adresse permet d'identifier la ou les stations secondaires impliquées dans l'échange de trames. Il existe deux formats possibles pour identifier l'adresse :

- soit le format de base qui occupe un octet,

- soit le format étendu. Dans ce cas, le format fait l'objet d'un accord préalable entre les différentes stations et le dernier élément binaire de chaque octet d'adresse indique s'il est le dernier ou pas.

Dans son application à TRANSPAC, le champ adresse adopte le format de base et il ne peut prendre que deux valeurs comme l'indique la figure 4.4.

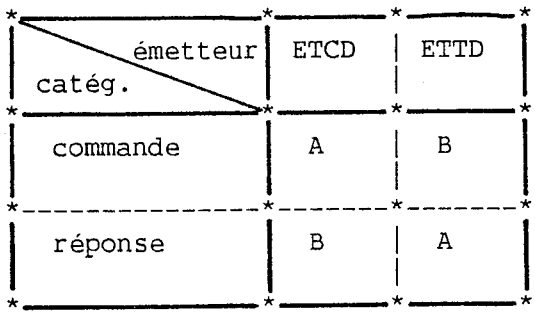


figure 4.4: Contenu du champ adresse.

Remarque:
les trames dont le champ adresse est différent de A ou B seront ignorées.

4.3.3. Le champ de commande.

Le contenu du champ de commande varie selon la catégorie de la trame émise (voir figure 4.5). Nous trouvons trois catégories :

- les trames d'information, notées (I), qui sont destinées à véhiculer de l'information.

- les trames de supervision numérotées, notées (S).

- les trames non numérotées (U) qui servent à exécuter des fonctions de commande de liaison.

élt binaires type de trame	1	2	3	4	5	6	7	8
trame (I)	0	N(S)			P/F	N(R)		
trame (S)	1	0	S	S	P/F	N(R)		
trame (U)	1	1	M	M	P/F	M	M	M

N(S) = numéro de séquence en émission
N(R) = numéro de séquence en réception
S = fonction de supervision
M = fonction de modification
P/F = invitation à émettre si commande
= fin si réponse

Figure 4.5: Format du champ de commande.

REMARQUES :

[a] L'élément binaire P/F (Poll/Final) a deux significations suivant que la trame provient d'une station primaire ou secondaire :

- le bit P (Poll) est utilisé par la station primaire dans une trame de commande pour solliciter une réponse de la station secondaire.
- le bit F (Final) sert à la station secondaire pour indiquer que la trame émise est la dernière concernant la réponse en court.

Ce bit P/F est très important dans HDLC car il permet de contrôler le bon enchainement des commandes et des réponses.

[b] Les bits M de modification figurant dans les trames (U) permettent de définir jusqu'à 32 commandes ou réponses supplémentaires en fonction de la classe de procédure.

[c] Le champ de contrôle peut avoir un format étendu à deux octets, ce qui autorise l'emploi du modulo 128 pour les numéros de séquence en émission et en réception.

4.3.4. Le champ d'information.

Il n'existe pas de restriction relative au codage du champ d'information, toutefois sa longueur maximum peut être négociée pendant la demande de connexion.

4.3.5. La séquence de contrôle de trame (FCS)

(=Frame Check Sequence)

Cette zone appelée également code de redondance cyclique permet de détecter des erreurs de transmission. Le FCS est calculé pendant l'émission de la trame en fonction du contenu de celle-ci puis recalculé en réception pour être comparé au code recu. Une différence entre les deux indique une erreur de transmission.

4.3.6. La transparence.

Pour supprimer toute ambiguité sur l'information contenue entre deux fanions, il est nécessaire de mettre en œuvre un mécanisme de transparence. Il consiste à insérer un zéro après chaque séquence de cinq éléments binaires à "1" pendant l'émission. Le récepteur doit examiner le contenu de la trame et éliminer tout élément "0" qui suit une séquence de cinq "1" consécutifs.

4.3.7. Les séquences réservées.

En plus du champ "fanion", il existe deux séquences d'éléments binaires réservées ayant une signification particulière :

- la séquence "abort" qui est une suite de 7 bits "1" consécutifs. Elle s'utilise pour terminer prématurément l'émission d'une trame (par exemple pour laisser la liaison à une trame plus prioritaire).

- la séquence "idle" ou repos constituée de 15 bits "1" consécutifs indique un état inactif de la liaison. Cette séquence peut indiquer un incident au niveau physique si elle survient alors qu'une séquence "fanion"

est attendue.

4.4. Eléments de procédure HDLC.

4.4.1. Numérotation et acquittement.

Chaque trame d'information émise est numérotée en séquence modulo 8 (champ N(S)). Ce numéro est vérifié par le récepteur qui gère à cet effet une variable d'état de réception (V(R)). Si le numéro de séquence est correcte, le récepteur "acquitte" cette trame au moyen du champ N(R) auquel il affecte la valeur N(S) + 1. Le champ N(R) peut être transmis

- soit par une trame d'information
- soit par une trame de supervision.

La réception d'un champ N(R) indique que toutes les trames émises ont bien été recues par le destinataire jusqu'à N(R) - 1.

4.4.2. Contrôle de flux.

La procédure HDLC utilise le mécanisme de la fenêtre CORN[81] pour contrôler le flux de transmission. Par ce mécanisme, l'émetteur dispose d'un "crédit" égale au nombre maximum de trames en cours d'acheminement (non acquitées) à un instant donné. La fenêtre de l'émetteur ou intervalle de numérotation a pour origine le numéro de la plus ancienne trame non acquittée et pour largeur la valeur courante du crédit. En réception, il n'y a pas d'anticipation.

4.4.3. Les principales Commandes et Réponses.

Les formats des commandes et réponses sont récapitulées en annexe 1.

4.4.3.1. La commande d'information (notée (I)).

Sa fonction est de transmettre sur la liaison de données des trames numérotées séquentiellement qui contiennent un champ d'information pour le niveau supérieur.

4.4.3.2. Commande et réponse prêt-à-recevoir ou (RR).

C'est une trame de supervision qui peut jouer deux rôles :

- indiquer un état prêt à recevoir une nouvelle trame (I)
- accuser réception des trames (I) recues.

4.4.3.3. Commande et réponse rejet (REJ).

La trame (REJ) permet de demander la retransmission des trames (I) à partir du numéro de séquence N(R). Parallèlement, la trame (REJ) accuse réception des trames jusqu'à N(R) - 1 inclus.

La condition d'exception "rejet" est annulée à la réception d'une trame (I) dont la séquence est égale au numéro N(R) demandé par la trame (REJ).

4.4.3.4. Commande et réponse non-prêt-à-recevoir (RNR).

Cette trame de supervision indique un "état d'occupation", c'est à dire que la station est dans l'incapacité temporaire de recevoir de nouvelles trames (I). Cela peut se produire par exemple si elle ne dispose plus de zone de mémoire suffisante pour stocker l'information à venir.

L'envoi de cette trame a pour effet de "bloquer" l'émission de la station correspondante. Dans l'état occupé, la station n'acquitte plus aucune trame d'information. Elle peut annuler cet état par l'émission de trame (UA), (RR), (REJ) ou (SABM) lorsque les conditions le permettent.

4.4.3.5. Commande de mise en mode asynchrone symétrique (SABM).

Il s'agit d'une trame non numérotée qui est utilisée pour placer la station appelée en phase de transfert d'information, soit pendant la connexion, soit pour réinitialiser la liaison. La trame (SABM) doit être confirmée par le correspondant au moyen d'une trame (UA).

4.4.3.6. Commande de déconnexion (DISC).

Cette commande permet de demander la fin du mode opérationnel qui était établi. Elle indique que l'émetteur du (DISC) suspend son fonctionnement. De même que (SABM), elle doit être confirmée par une trame (UA).

4.4.3.7. Réponse d'accusé de réception (UA).

Comme nous l'avons déja vu cette trame non numérotée permet d'accuser réception des commandes (SABM) et (DISC).

4.4.3.8. Réponse en mode déconnecté (DM).

La trame (DM) signale au correspondant un état logiquement déconnecté de la liaison.

4.4.3.9. Réponse de rejet de commande (CMDR) ou rejet de trame (FRMR).

Ces deux trames de réponse sont utilisées pour signaler une condition d'erreur sur le contenu ou la longueur d'une trame de commande ou d'information. De plus elles contiennent un champ d'information de trois octets pouvant servir à préciser la raison du rejet.

4.5. Description des procédures.

Nous avons vu en 4.1.2 qu'il existe tois modes de fonctionnement possible avec la procédure HDLC :

- NRM = La station secondaire ne peut émettre que si elle a reçu au préalable une autorisation (bit P=1).
- ARM = Chaque station cumule les fonctions primaires et secondaires.
- ABM = Une station combinée peut initialiser une transmission.

Notre objectif étant la mise en œuvre d'HDLC pour un raccordement au réseau TRANSPAC, nous ne retiendrons que le mode ABM recommandé par TRANSPAC sous le nom de procédure LAPB.

Dans la durée de vie d'une liaison de données, nous distinguerons trois phases : l'établissement et la déconnexion de la liaison ainsi que la phase de transfert de l'information.

4.5.1. L'établissement de la liaison.

Le protocole de connexion défini par HDLC est très simple et se fait en deux temps :

- premier temps, il suffit à l'ETTD d'émettre une commande (SABM).
- deuxième temps : La connexion sera effective après la réception de l'acquittement (UA).

A cet instant, l'ETTD initialisera ses variables à zéro et il sera prêt à émettre ou à recevoir des trames d'information.

Si l'ETTD ne recoit pas l'accusé de réception avant l'échéance d'un délais de garde (T2) prévu au départ, il doit réémettre (SABM). Après un nombre (N2) de retransmissions infructueuses, l'ETTD considèrera que la liaison est indisponible.

Remarque: T2 et N2 sont des paramètres du système.

4.5.2. Phase de transfert d'information.

Pendant cette phase, qui suit immédiatement une connexion réussie, l'ETTD peut émettre et recevoir des trames d'information ou de supervision et il doit réagir aux incidents.

L'émission d'une trame (I) se fait conformément au principe de numérotation décrit en section 4.4.1 et en respectant le facteur d'anticipation (K) négocié. Si l'ETTD n'est pas en attente d'acquittement de trame, il doit armer un temporisateur (T2) qui le préviendra en cas de dépassement du délais de garde. Ce délais correspond a la durée maximum d'acheminement toléré d'une trame.

Dans le cas du dépassement, la trame sera réémise dans la limite du nombre de retransmission autorisé (N2).

En réception, l'ETTD doit vérifier la validité des trames qu'il recoit (champ FCS, longueur, séquence, etc...). Si une trame (I) est correcte, elle sera acquittée par la procédure décrite en section 4.4.1 mais certains incidents peuvent venir perturber le déroulement normal des échanges de trames, notamment :

- la réception d'une trame non valide
- un état d'occupation de la liaison
- l'échéance d'un temporisateur.

Les paragraphes suivants décrivent la réponse de L'ETTD aux différents types d'incident.

4.5.2.1. Réception de trames incorrectes.

Plusieurs cas sont envisageables :

- le contenu d'un champ n'est pas valide (FCS, adresse ou commande). Dans ce cas, la trame est simplement ignorée. C'est l'absence d'acquittement qui fera réagir la station émettrice de cette trame.
- le format de la trame est incorrecte (trame trop longue ou trop courte). L'ETTD peut émettre une réponse de rejet de commande ou de trame suivant le cas.

- si l'erreur provient d'un mauvais séquencement, le champ d'information est ignoré, l'ETTD émet une réponse (REJ) puis se place dans l'état d'exception "rejet" (voir la section 4.4.3.3).

- en cas d'erreur liée à une désynchronisation des deux stations, par exemple à la réception d'un champ N(R) non compris dans l'intervalle [dernier N(R) recu, prochain numéro de séquence à émettre], l'ETTD demandera une réinitialisation de la liaison par l'émission de (SABM).
  Remarque: la réinitialisation provoque la remise à zéro de toutes les variables internes ce qui peut conduire à la perte de trames en cours d'acheminement mais jamais à une duplication.

- enfin certaines erreurs irrécupérables peuvent entraîner un abandon de la liaison. C'est le cas lorsqu'on dépasse le nombre de retransmissions permises.

4.5.2.2. Etat occupé du récepteur.

Lorsque l'ETTD se trouve dans l'incapacité de recevoir de nouvelles trames (I), il bloque l'émission de son correspondant par l'envoi d'une trame (RNR) (voir section 4.4.3.4). Il peut revenir à l'état normal quand la cause de l'état d'occupation a disparu (par exemple après libération d'espace de mémoire).

4.5.2.3. Echéance d'un temporisateur.

Quand une trame (I) n'est pas acquittée dans les délais impartis, la station réémet cette trame avec le champ P=1 et se place dans l'état "réveil". Dans cet état, l'ETTD ne pourra pas émettre d'autres trames et seule la réception d'une trame de supervision avec le champ F=1 pourra le sortir de cet état.

4.5.3. Déconnexion de la liaison.

Pendant la phase de transfert d'information une station peut demander l'arrêt de la liaison par l'émission d'une trame (DISC) (section 4.4.3.6). La déconnexion est confirmé par la réception

d'une trame (UA).

4.6. Représentation du protocole.

Il existe plusieurs possibilités pour décrire un protocole: types abstraits, réseau de pétri, langages algorithmiques etc... Pour des raisons pratiques (cohérence avec le chapitre suivant), nous présentons la description très générale du protocole HDLC sous forme de diagrammes d'état tel qu'il est mis en œuvre dans TRANSPAC TRAN[83].

Nous trouverons les phases de connexion/déconnexion dans la figure 4.6, la phase d'émission de trames I avec la figure 4.7 et la phase de réception de trames I dans la figure 4.8.

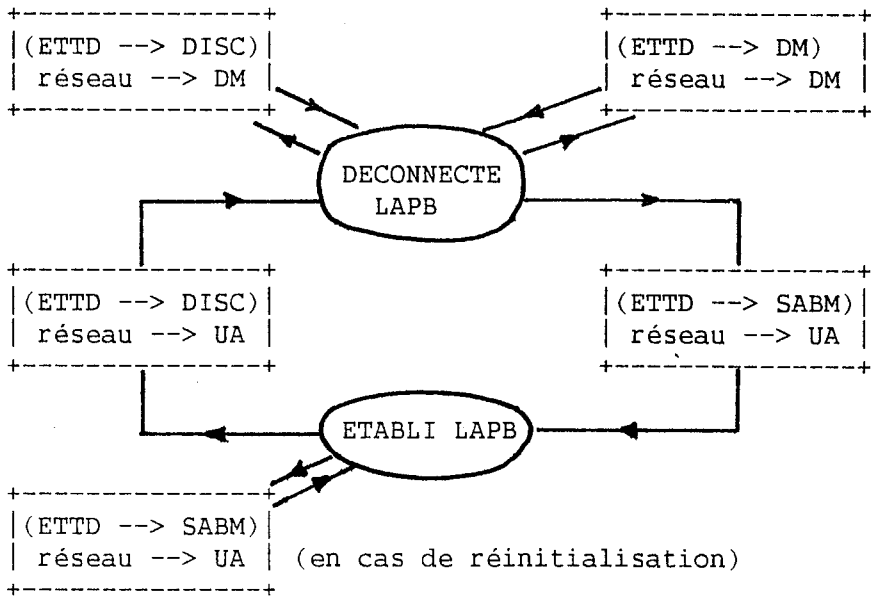


Figure 4.6 : phase de connexion et déconnexion.

Diagramme applicable à l'état "ETABLI LAPB"

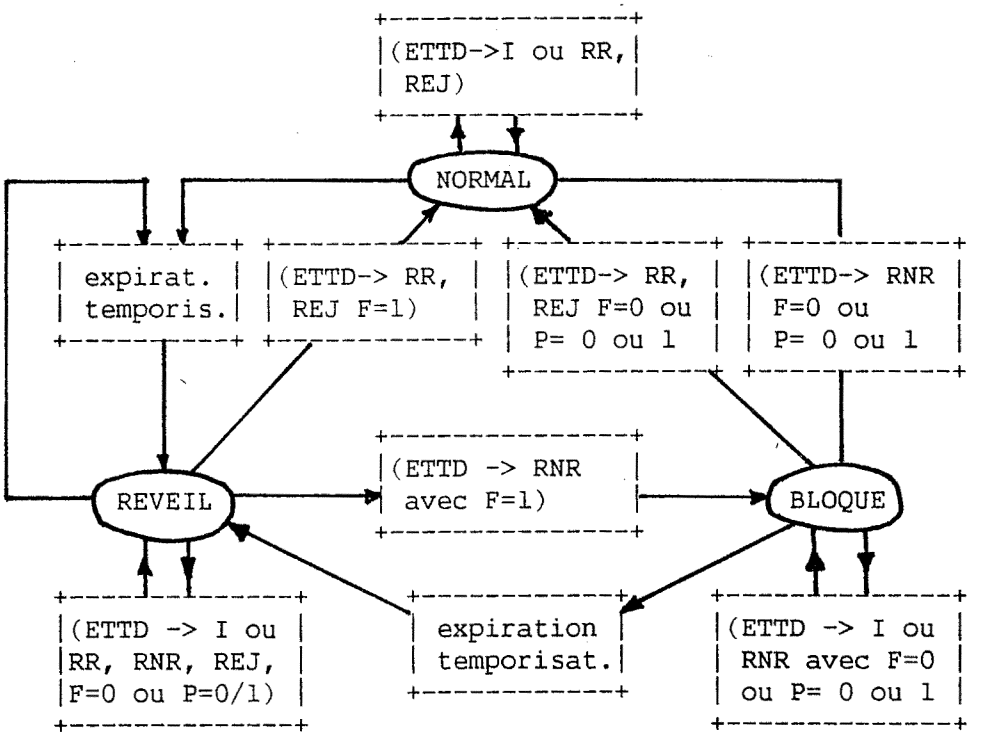
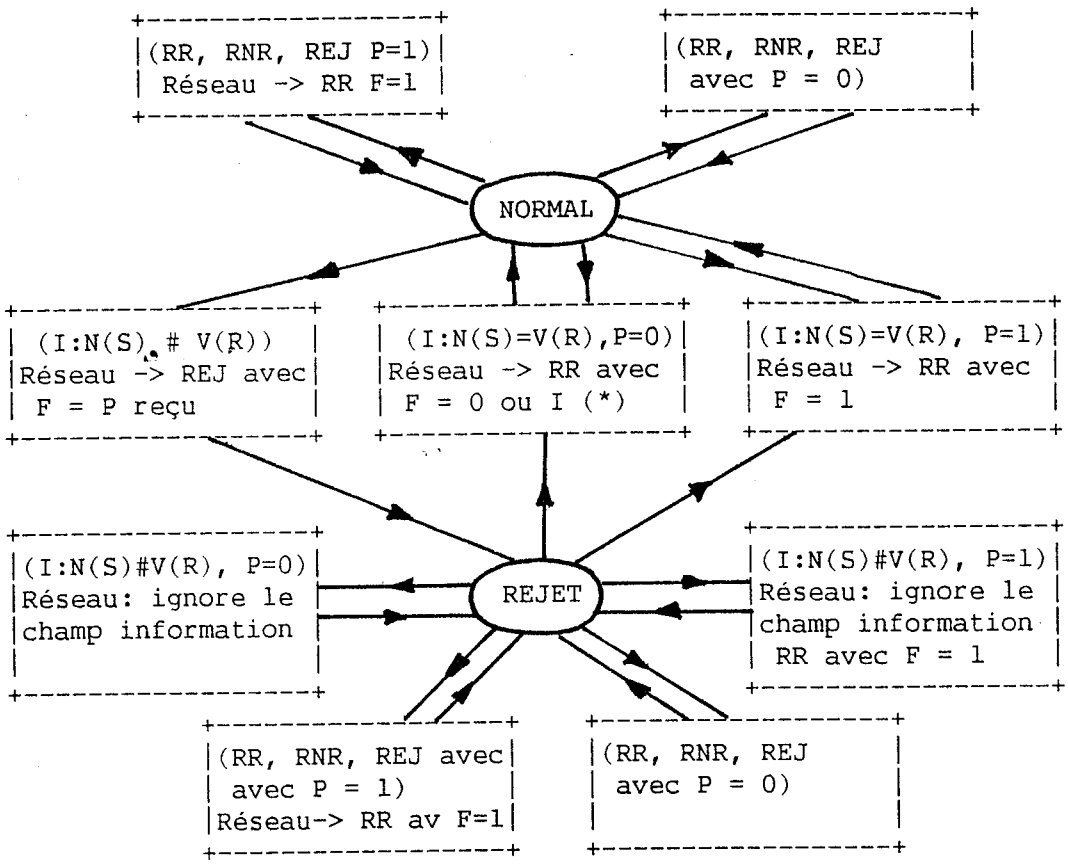


Figure 4.7 : phase d'émission de trames I.



(*) le réseau transmet de préférence une trame I.

Figure 4.8 : phase de réception de trames I.

5. Implémentation du protocole H.D.L.C

5.1. Choix d'une architecture.

5.1.1. Problème posé.

L'objectif du travail à réaliser consiste à fournir à l'utilisateur un service d'échange d'informations avec un réseau du type TRANSPAC à l'aide de fonctions standards dans un environnement donné avec comme contraintes:

- Le système d'exploitation UNIX.
- L'utilisation du coupleur MLCP.
- Les lignes de communication synchrones à moyenne vitesse.
- Le respect de la norme X25.

Dans ce chapitre nous ne retiendrons que les aspects relatifs au niveau 2 du modèle ISO, c'est à dire le niveau liaison de données avec le protocole HDLC. La figure 5.1 représente l'ensemble des éléments constitutifs du problème.

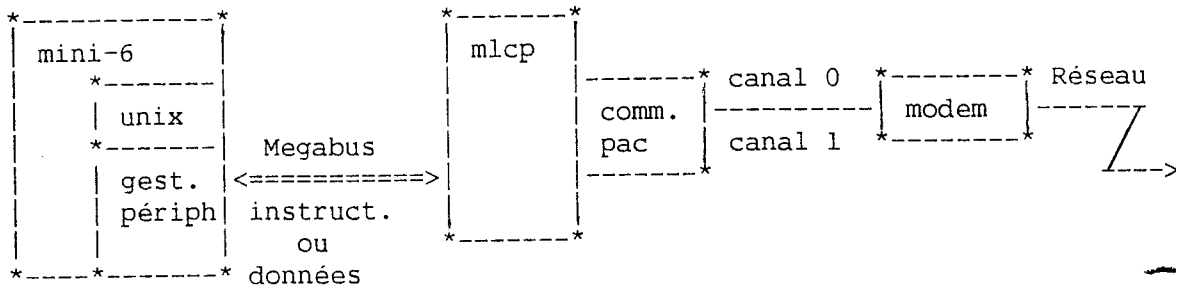


Figure 5.1: Eléments de la liaison de données

5.1.2. Les solutions possibles.

En tenant compte du fait que le coupleur mlcp est programmable, nous pouvons envisager au moins trois types d'architecture possible pour la mise en œuvre du protocole HDLC:

[a] Soit gérer tout le protocole au niveau du mini-6, c'est à dire dans le programme "driver" ligne HDLC. Dans ce cas, le coupleur voit son utilisation réduite au simple transfert de caractères entre la ligne de communication et la mémoire du mini-6.

[b] Soit gérer le protocole HDLC au niveau du coupleur. Ce système revient à préparer des zones tampons en mémoire centrale pour stocker l'information à transmettre et à communiquer au coupleur des primitives globales comme: démarrer le protocole ou initialiser un transfert d'information.

[c] La troizième possibilité est une solution mixte qui consiste à répartir les différentes fonctions de gestion du protocole entre le mini-6 et le coupleur.

La première solution présente surtout l'avantage de la simplicité :

+ simplicité de codage avec le langage "C"

+ réduction du programme du coupleur à sa plus simple expression (transfert caractère par caractère)

+ simplification de l'interface mini-6 / coupleur: commande d'émission et indication de réception de trame.

mais les inconvénients qu'elle comporte sont difficilement supportables :

- surcharge du processeur du mini-6 (longueur du code et nombre d'interruptions)

- sous-emploi du processeur du coupleur mlcp

- augmentation importante de la taille du noyau UNIX.

La deuxième possibilité avec la gestion du protocole dans le coupleur permet de supprimer la plupart des inconvénients que nous venons d'énumérer. De plus, elle s'intègre bien dans une architecture en couches car elle regroupe sur un même support (le

coupleur) toutes les fonctionnalités d'une couche du modèle.

Malheureusement, sa mise en œuvre pose plusieurs problèmes dont certains sont bloquants du fait des fonctionnalités insuffisantes du coupleur comme l'absence de temporisateur (timer) ou le jeu d'instructions disponible trop réduit.

5.1.3. Le choix d'une architecture.

Comme bien souvent, c'est un compromis entre les deux extrèmes qui a été retenu avec la solutions mixte. Une des difficultés consistait à répartir au mieux les fonctions entre le mini-6 et le coupleur en préservant le plus grand nombre d'avantages inhérents à chacune des solutions.

La plus grande partie du protocole HDLC a donc été réalisée dans le coupleur (numérotation des trames, acquittement, émission des commandes) tandis que le programme "driver" ligne HDLC du mini-6 prend à sa charge la gestion des temporisateurs ainsi que les demandes de retransmission. Cette dissociation de la fonction de temporisation rend la synchronisation mini-6 / coupleur plus délicate mais elle n'a pas pu être évitée.

Dans les sections qui suivent, nous allons en premier lieu étudier les caractéristiques du programme chargé des entrées-sorties sur les lignes HDLC dans le mini-6 (ce programme est appelé "driver"), puis nous présenterons la mise en œuvre du protocole dans le coupleur mlcp.

5.2. Le programme driver pour les lignes HDLC.

5.2.1. Gestion des périphériques dans UNIX.

Nous avons vu dans le chapitre 3 que le système UNIX considère les périphériques comme des fichiers qu'il est possible d'ouvrir, lire, écrire ou fermer. La distinction entre fichiers ordinaires et périphériques s'opère dans la couche inférieure du noyau. Pour complèter ces remarques, il faut distinguer deux classes de périphériques dans UNIX :

- les périphériques en "mode bloc"

- les périphériques en "mode caractère"

La principale différence entre ces deux classes réside dans le fait que dans le premier cas les transferts mémoire/périphérique sont fait par blocs de 512 caractères. De plus, le mode bloc comporte une gestion complexe de zones tampon qui servent de relais entre les entrée-sortie logiques et physiques. Ces zones de mémoire intermédiaire, qui sont affectées sous le contrôle de listes chainées, permettent d'optimiser les accès physiques aux périphériques mais elles impliquent également l'exécution de code supplémentaire. C'est de cette manière que sont gérés les disques et bandes magnétiques.

La classe "mode caractère" n'utilise pas les mécanismes de zone "tampon", les entrée-sortie sont assurées caractère par caractère entre la mémoire centrale et le périphérique. Dans cette classe nous trouvons tous les périphériques dits "lents" ainsi que d'autres qui travaillent habituellement par bloc mais pour lesquels on veut éviter le système des zones tampon. C'est également dans cette classe que nous trouvons les lignes de communication.

Dans UNIX, un périphérique est identifié par le système grâce à sa classe et à un numéro composé de deux parties:

- le "majeur" qui désigne le type de périphérique, c'est à dire les procédures qui vont le gérer (programme driver)

- le "mineur" qui est un identificateur à l'intérieur d'un type donné.

Dans le chapitre 3 nous avons également remarqué qu'une "table de configuration" est associée à chacune des deux classes de périphérique. Cette table comporte une entrée par type de périphérique avec les adresses des procédures de gestion correspondantes.

5.2.2. Structure d'un programme driver.

Pour simplifier la présentation, nous ne retiendrons plus que la classe de périphériques en mode caractère dans laquelle nous allons insérer notre driver HDLC. Un driver peut être considéré comme une entrée dans la table de configuration (par le numéro de majeur) associée à un ensemble de procédures indépendantes qui assurent la communication avec le type de

périphérique concerné.

Ces procédures se situent à deux niveaux :

- celles qui qui sont appelées par le système de gestion des entrée-sortie : ouverture, fermeture, lecture ou écriture et opérations de contrôle

- celles qui sont éxécutées lors de la prise en compte d'une interruption en provenance d'un périphérique. Ces dernières sont liées à un niveau d'interruption (celui du périphérique qu'elles servent). Ces procédures sont appelées des "handlers" d'interruption et nous trouvons des "handlers" pour les interruptions d'entrée et des "handlers" pour les interruptions de sortie.

5.2.3. Insertion d'un nouveau driver.

Pour schématiser, nous pouvons dire que l'ajout d'un nouveau "driver" consiste à créer une nouvelle entrée dans la table de configuration située dans le programme chargé de définir l'environnement matériel du système (config.c). Cette entrée fait référence aux procédures de gestion d'entrée-sortie et d'interruption qu'il faut créer dans un programme que nous avons nommé "cl.c". Ce programme fait lui-même appel à des modules d'entrée-sortie physiques écrits en assembleur mini-6 dans un fichier "hdlc.s".

Chacun des fichiers sources composant le noyau est compilé séparément et ensuite l'édition de liens est globale pour générer un fichier exécutable "/unix" qui est en fait le nouveau système. Ce fichier "/unix" sera copié en mémoire centrale pendant le chargement du système (bootstrap).

5.3. Le coupleur MLCP.

5.3.1. Généralités.

Le MLCP CII[81] (Multiline Communications Processor) est un processeur qui permet de gérer plusieurs lignes de communication par l'intermédiaire d'interface de ligne appelées

"communications-Pac". Il est programmable et il se connecte sur le Megabus du mini-6 comme un autre périphérique. Le MLCP constitue la partie commune à plusieurs lignes de communication synchrones ou asynchrones, tandis que la logique spécifique à chacune d'elles se situe dans le "Communications-Pac" adapté.

Un coupleur MLCP peut supporter jusqu'à huit lignes asynchrononnes ou quatre lignes synchrones HDLC mais les deux types de ligne peuvent coexister simultanément. La figure 5.2 nous donne un aperçu de l'architecture du mlcp.

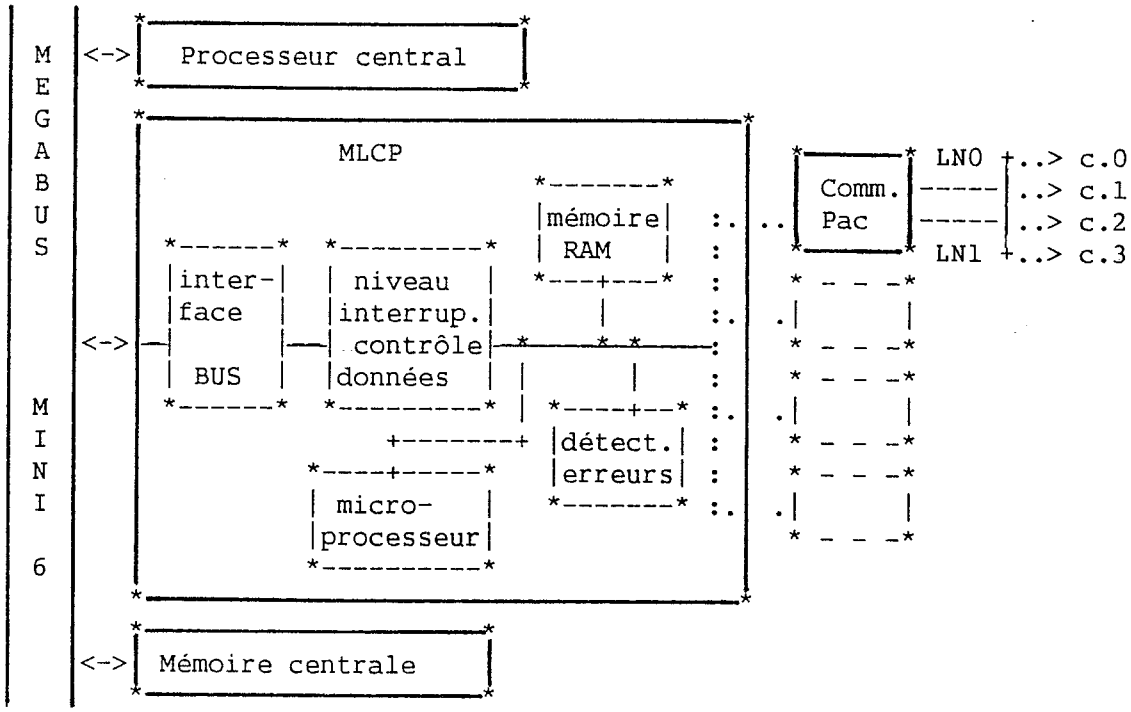


Figure 5.2 : Architecture du MLCP

Le Communications-Pac se charge de la conversion série-parallèle et inversement pour les caractères reçus ou transmis.

5.3.2. Fonctionnement du MLCP.

L'information échangée entre le mini-6 et le MLCP transite par une zone en mémoire centrale appelée CDB (Communications Data Blocks). Cet espace est chargé par le mini-6 et lu par le MLCP

pour une émission ou chargé par le MLCP en cas de réception.). Le processeur du MLCP gère l'adressage de cette zone grâce à des listes circulaires chainées appelées CCB (Communications Control Blocks). Les CCBs sont initialisées dans la RAM du MLCP par le programme driver situé en mémoire centrale à l'aide d'instructions d'entrée-sortie particulières puis elles sont mises à jour par le programme canal du MLCP.

Pendant qu'une donnée transite du Megabus vers le Communications-Pac ou vice versa, le programme canal du MLCP appelé CCP (Channel Control Program) exerce un contrôle sur le contenu et le format du flot de données. Un programme canal doit pouvoir communiquer aussi bien avec le programme driver en mémoire centrale qu'avec le programme de gestion de l'autre canal de la ligne hdlc. Pour cela, le CCP dispose de deux moyens:

- Le premier consiste à générer des interruptions qui vont interrompre l'exécution du programme en mémoire centrale.

- Le second permet au CCP d'écrire dans des zones mémoire de la RAM appelées LCT (Line Control Table) qui sont accessibles au programme driver ligne comme à l'autre programme canal.

Le fonctionnement du MLCP est contrôlé par un logiciel de base microprogrammé qui assure entre autre :

- les sauvegardes et restaurations de contexte (registre, indicateurs d'état)

- les appels et retours de procédures

- le partage des ressources

auquel s'ajoute une logique cablée spécifique à certains traitements comme :

- le contrôle de parité

- le calcul automatique du champ FCS d'une trame.

5.3.2.1. La segmentation de la mémoire du MLCP (RAM).

Dans ce qui suit nous allons examiner de plus près le contenu des différentes zones de la mémoire du coupleur : LCT, CCP et CCB.

5.3.2.2. La table de contrôle de ligne ou LCT.

Cette table est une zone de 64 octets réservée pour chaque ligne de communication et dont l' accès est interdit aux autres lignes. Une LCT est logiquement divisée en deux parties de 32 octets, la première est affectée au canal de réception alors que la seconde est allouée au canal d'émission, mais il faut noter que les deux programmes canaux d'une même ligne peuvent accèder à toutes les LCTs de cette ligne. Parmis les LCTs, certaines peuvent être modifiées par les programmes alors que les autres ont un usage réservé au logiciel de base.

Nous trouvons plusieurs types d'information dans les LCTs :

- des informations d'entrée pour le logiciel de base comme :
  - la configuration des caractères
  - le pointeur d'instruction
  - les conditions d'erreur
- des zones de travail internes du processeur :
  - sauvegarde de contexte
  - pointeurs de CCBs
- des zones de travail pour le programmeur.

5.3.2.3. Les programmes de contrôle des canaux ou CCPs.

Un CCP a pour objectif de contrôler le flux de données entre le Communications-Pac et la zone tampon (CDB) en mémoire centrale, caractère par caractère, pour le compte d'un canal, si bien qu'une ligne est gèrée par deux CCPs. Les CCPs sont des

programmes réentrants, c'est à dire que le code peut être partagé par plusieurs lignes, les données ou contexte étant stockés dans des LCTs ou CCBs spécifiques à chaque ligne.

Le déroulement d'un CCP est piloté par le programme driver qui dispose d'instructions spéciales d'entrée-sortie pour lancer ou arrêter l'exécution d'un CCP. Le processeur du MLCP est alloué tour à tour à chacun des CCPs, qui devient actif, par le logiciel de base.

Les CCPs disposent d'un faible jeu d'instructions peu adaptées à la programmation d'un protocole comme HDLC, par exemple, il n'existe pas d'opérateurs arithmétiques. Une simple opération comme A := A+1 modulo 8 doit être programmée à l'aide d'opérateurs logiques et de masques. Comme nous l'avons déja signalé, les CCPs ne comportent pas non plus de ressources temporisateur et leur mémoire de travail est très réduite, quelques octets.

5.3.2.4. Les blocs de contrôle ou CCBs.

Les CCBs sont des zones utilisées par les CCPs comme interface avec les CDBs. Elles sont gèrées sous forme de listes circulaires et elles contiennent les informations nécessaires au transfert des données : adresse en mémoire centrale, longueur de la zone, état du transfert. Les CCBs sont initialisés par le programme driver avec des instructions d'entrée-sortie spécifiques (iold) puis maintenus par le MLCP pendant les transferts. Un CCB est composé de quatre champs :

- ADRESSE : adresse de la zone tampon d'entrée-sortie. Elle est incrémentée après chaque transfert de caractère.

- RANGEMENT : nombre d'octets à transférer ou à recevoir. Son contenu est décrémenté après chaque transfert.

- CONTROLE : contient les informations de contrôle du transfert comme le masque d'interruption ou l'indicateur de fin de bloc.

- ETAT : ce champ contient tous les indicateurs d'état du transfert comme les erreurs dues au débit (overrun) ou à l'adressage.

La mise à jour des pointeurs de la liste circulaire des CCBs est assurée par le programme canal avec l'instruction gnb (Get Next Block) qui fait passer au CCB suivant, renseigne les indicateurs d'état et de contrôle puis interrompt éventuellement le mini-6.

5.3.3. Les Communications-Pac synchrones.

Ils contiennent l'interface cablée entre le coupleur et la ligne de communication HDLC. Ils assurent la sérialisation des bits d'information lors du transfert d'information et la conversion série-parallèle lors de la réception grâce à une mémoire tampon (registre). Ils commandent également les circuits V24 vers le modem.

La synchronisation coupleur - Communications-Pac est réalisée par les registres de ligne (LR) et les instructions d'entrée-sortie (send, out, recv, in). Certaines opérations liées à la procédure HDLC sont éxécutées directement par les Communications-Pac comme la transparence (insertion-suppression de zéro) ou la reconnaissance de séquences réservées "fanions" ou "abort".

5.3.4. Chargement du coupleur MLCP.

La première opération à effectuer pour la mise en ordre de marche du MLCP consiste à écrire le code des programmes CCPs dans la mémoire du coupleur. Pour cela nous avons utilisé un driver dans le noyau d'UNIX qui considère la mémoire du MLCP comme un périphérique.

Un programme de chargement de niveau utilisateur (hdlcboot) permet de copier le code du CCP des lignes HDLC à une adresse particulière (1024) de la RAM. De cette manière, la première partie de la mémoire peut recevoir les CCPs destinés à la gestion des lignes asynchrones.

La figure 5.3 nous montre le découpage de la mémoire RAM du MLCP.

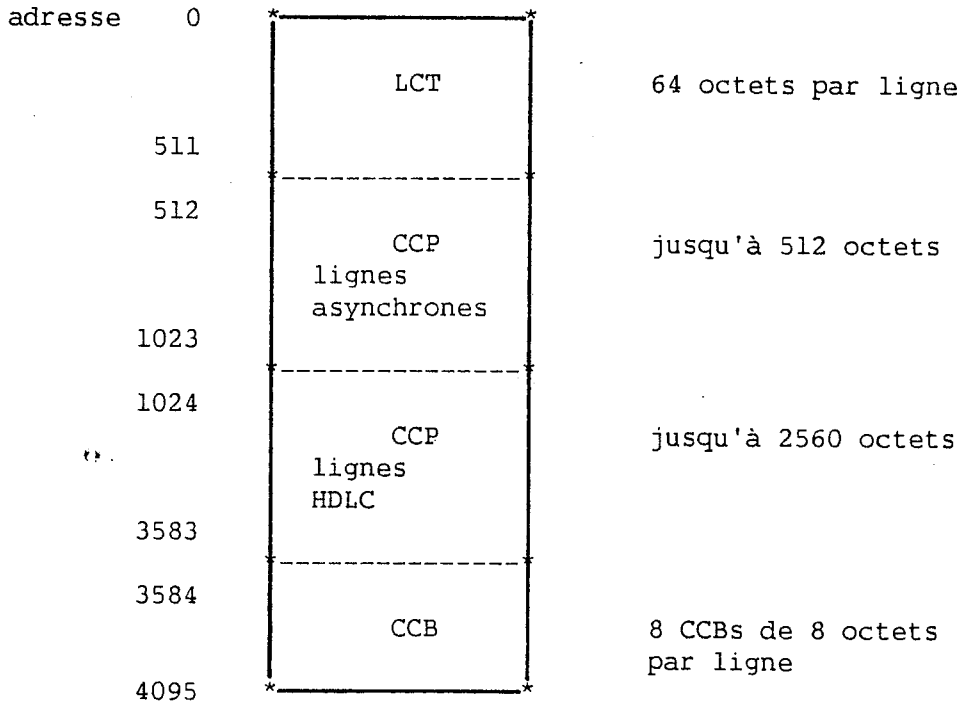


Figure 5.3 : structure de la mémoire du MLCP

5.4. Implémentation du protocole HDLC.

5.4.1. Les types d'accès proposés.

La procédure HDLC telle que nous l'avons mise en œuvre offre deux types de service :

- des primitives d'accès destinées au niveau immédiatement supérieur dans la norme X25 (paquet).

- des primitives d'accès pour le niveau utilisateur, c'est à dire extérieur au noyau Unix. L'annexe II présente les pages du manuel utilisateur pour l'exploitation du driver HDLC.

L'interface utilisateur permet d'accéder aux lignes HDLC par les appels système standards (system call) d'entrée-sortie:

open() Réservation d'une ligne et lancement du protocole.

read() Lecture bloquante d'une séquence de longueur variable.

write() Ecriture bloquante d'une séquence de longueur variable.

ioctl() Commandes de contrôle (essentiellement statistiques).

close() Arrêt du protocole et restitution de la ligne.

Il est construit dans une couche de logiciel du driver qui utilise en interne les procédures de l' interface x25 que nous allons examiner.

5.4.2. Les primitives d'accès du driver HDLC au niveau x25.

Ces primitives ont été définies en relation avec les autres participants au projet EIES pour assurer une certaine homogénéité à l'ensemble du système. Chacune d'elles est supportée par un bloc écrit en langage "C" dans le programme driver. Nous trouvons ainsi :

clattach : réservation d'une ligne HDLC.

Cette fonction initialise une structure associée à la ligne de communication au moyen du numéro de canal physique. Lorsqu'elle est appelée par l'appel système open(), elle effectue également la réservation des tampons d'entrée-sortie des trames grâce au système de gestion mémoire développé pour ce projet. Ce module gère la négociation des paramètres associés à la ligne.

clstart : démarrage du protocole HDLC.

Le démarrage se fait en trois étapes :

- copie du contenu des LCTs dans la mémoire du MLCP
- activation du programme canal de réception qui sera en "écoute" sur la ligne
- activation du programme canal d'émission dans sa phase de connexion.

Après l'appel de clstart, le processus est mis en "sommeil" jusqu'à ce qu'il soit "réveillé" par une interruption en provenance du coupleur indiquant la fin. Quand la connexion est réussie clstart retourne un "descripteur de ligne".

clstop : arrêt du protocole HDLC.

Avant d'arrêter le coupleur, cette procédure vérifie qu'aucune trame n'est en cours d'acheminement (trame émise non encore acquittée). La déconnexion se fait par l'activation du CCP d'émission en phase d'arrêt avec mise en sommeil du processus. La déconnexion sera effective après réception d'une interruption du MLCP qui réveille le processus appelant.

REMARQUE: les procédures clstart() et clstop() gèrent un temporisateur qui permet de réactiver les phases de connexion et déconnexion du MLCP en cas d'échec.

cldetach : libération d'une ligne HDLC.

Elle libère la structure attachée à la ligne par clattach.

clrecv : initialisation de la réception d'une trame.

Cette procédure permet de fournir une nouvelle adresse et un nombre de caractères au mécanisme de gestion des mémoires tampons. Dans le cas d'un appel par un processus utilisateur, elle examine la liste des trames recues pour délivrer la plus ancienne. Si aucune trame n'est disponible le processus est mis en sommeil et il sera réveillé par l'interruption indiquant la réception d'une trame d'information.

clsend : initialisation de l'émission d'une trame.

Cette procédure est symétrique à clrecv mais elle gère en plus le mécanisme de la fenêtre (trames en anticipation). Clrecv et clsend commandent l'initialisation des CCBs dans le MLCP (chargement de l'adresse et de la longueur de la trame).

REMARQUE: c'est le niveau paquet d'X25 qui est responsable de l'approvisionnement en zones tampons pour l'émission et la réception de trames d'information.

clstat : statistiques du niveau ligne.

Ce module est chargé de délivrer les statistiques maintenues par les autres procédures du driver HDLC dans la structure ligne.

clclear : libération logique d'une ligne sur incident.

Son rôle consiste à réinitialiser la structure associée à la ligne lorsque cette dernière reste bloquée après un incident.

Le driver HDLC contient également deux procédures de gestion des interruptions qui sont appelées directement par le noyau Unix lorsqu'une interruption émise par le MLCP est prise en compte par l'unité centrale du mini-6, suivant son niveau de priorité. Pour savoir quelle est la cause de l'interruption, le driver va lire le contenu de LCTs particulières dans la mémoire du MLCP.

clitsend : interruption en émission.

Le driver peut recevoir des interruptions en émission dans deux cas :

- demande de réémission d'une trame suite à l'expiration d'un temporisateur. Cet appel provoque l'incrémentation du nombre de retransmission, le rechargement des CCBs à partir de la trame à réémettre et la relance du CCP d'émission.

- demande d'armement d'un temporisateur après émission d'une trame.

clitrcv : interruption en réception.

Les causes d'interruption en réception sont plus nombreuses, nous trouvons :

- la libération d'un CCB d'émission qui entraine l'activation d'un nouveau CCB d'émission.

- la libération d'un CCB de réception provoque l'activation d'un nouveau CCB de réception et l'appel au niveau supérieur.

- l'acquittement suite à :

  - une connexion avec mise à jour de la variable

d'état et réveil du processus appelant.

- une déconnexion avec la même réaction que ci-dessus.

- l'envoi d'une trame d'information. Dans ce cas, le driver prévient le niveau supérieur qu'une trame émise a été correctement reçue.

- la demande de déconnexion du réseau nécessite la mise à jour de la variable d'état ainsi que l'appel au niveau paquet.

- un ordre de réinitialisation de la ligne provoque la réinitialisation de toutes les variables de la ligne, la revalidation de tous les CCBs, un appel au niveau paquet et la relance du CCP d'émission.

- la demande de réémission d'une trame d'information après un rejet est traitée par la réactivation des CCBs d'émission à partir de la trame rejetée et la relance du CCP d'émission sur un point de reprise particulier.

- une demande d'armement ou de désarmement d'un temporisateur.

Le driver peut communiquer au MLCP l'expiration d'un temporisateur en écrivant dans un mot de LCT particulièr.

5.4.3. Codage des programmes canaux du MLCP.

L'annexe III contient un organigramme des programmes canaux.

Nous avons déja vu qu'une ligne HDLC comporte deux canaux et que chacun d'eux est géré par un CCP particulier qui s'exécute indépendamment. Nous avons donc écrit un CCP de réception et un CCP d'émission.

5.4.3.1. Le programme canal de réception.

Ce programme étant le premier lancé lors de la connexion HDLC, c'est lui qui initialise les caractéristiques physiques de la liaison telles que :

- le champ de contrôle du FCS (voir 4.1.3.5).
- la configuration des caractères
- la mise en route logique du coupleur.

Il détecte ensuite les début et fin de trame grâce à l'identification des fanions. Lorsqu'une trame complète est reçue, elle est contrôlée par rapport à sa forme et à son contenu, par exemple les champs N(S) et N(R). Le champ de commande des trames valides est analysé pour choisir le traitement à effectuer parmi :

- trame d'information :

  Le contenu du champ d'information est stocké dans la zone destinataire en mémoire centrale (CDB). En réalité, les caractères sont transférés dans cette zone au fur et à mesure de leur réception car le MLCP ne dispose pas de mémoire tampon. La variable V(R) est incrémentée de un et une interruption est envoyée au mini-6 pour signaler la réception d'une trame d'information valide.

- trame de supervision :

  Dans ce cas le CCP réagit en fonction du protocole qui a été défini (chapitre 4). Cela peut se traduire par le passage dans un état particulier, par exemple l'état "réveil", ou par la demande d'émission d'une trame de supervision comme un acquittement "RR". Pour ce faire, les informations nécessaires sont communiquées soit au driver, soit au CCP d'émission par le positionnement d'éléments binaires prévus à cet effet dans les LCTs.

Lorsque la réponse à émettre est un acquittement de trame d'information, c'est le CCP d'émission qui choisira si il doit transmettre une trame "RR" ou une nouvelle trame d'information en accordant la priorité à cette dernière solution. Pour communiquer avec le driver, le CCP "charge" une LCT particulière (INTER) qui indiquera le type de message puis transmet une interruption sur le bus du mini-6. Notons qu'après le traitement complet d'une trame, le CCP de réception se place en "écoute" sur la ligne pour détecter le début d'une nouvelle trame.

5.4.3.2. Le programme canal d'émission.

Ce programme se décompose en trois phases:

- la phase de connexion.
- la phase de déconnexion.
- la phase de transfert d'information.

La phase de connexion est activée soit par le driver lors d'une mise en route de la ligne, soit par le CCP lui-même pour réaliser une réinitialisation de la ligne après détection d'une anomalie. Cette phase consiste à initialiser les variables dans les LCTs puis à émettre vers le réseau une trame "SABM" avant de mettre le CCP d'émission en sommeil (en attente d'une réponse).

La phase de déconnexion fonctionne exactement suivant le même principe mais avec émission d'une commande "DISC". Nous avons vu que les temporisateurs et les retransmissions étaient gérés par le driver HDLC, si bien que les phases de connexion et déconnexion peuvent être relancées plusieurs fois en cas de non-réponse.

Durant la phase d'échange d'informations, le CCP d'émission va boucler sur l'analyse de l'état de la ligne et du contenu de la LCT qui indique les trames commandes ou réponses à transmettre pour se dérouter ensuite sur la procédure correspondante.

REMARQUES :

Il faut savoir que le mini-6 ne mémorise pas les interruptions qu'il ne peut pas honorer immédiatement d'où un risque certain de perte malgré le stockage de quatre interruptions en attente effectué par le MLCP. Cette éventualité pourrait entrainer une désynchronisation entre le MLCP et le mini-6 en ce qui concerne les trames reçues où émises. Ce problème a été résolu en gérant des doubles cumuls de trames transmises une fois dans le driver et une fois dans le MLCP. La comparaison de ces deux cumuls permet de se synchroniser facilement à chaque interruption.

Par contre, il subsiste un risque de déclassement en se qui concerne l'ordre de communication des évènements, par exemple rejet d'une trame, car elle se fait par le positionnement d'éléments binaires dans un mot de LCT, ce qui ne permet pas de garantir la chronologie. Ce problème a été

minimisé en attachant un niveau de priorité à chaque type d'évènement, par exemple une demande de réinitialisation sera plus prioritaire qu'un armement de temporisateur. La meilleure solution consiste à stocker les évènement dans une pile de type "fifo" mais nous ne disposons pas de l'espace mémoire suffisant dans les LCTs pour le faire.

5.4.4. Mise au point et tests.

Dans le domaine des télétransmissions, la phase de mise au point est toujours l'une des plus délicates. Pour ce qui nous concerne, nous avons disposé des outils de test que nous voyons apparaitre dans la figure 5.4.

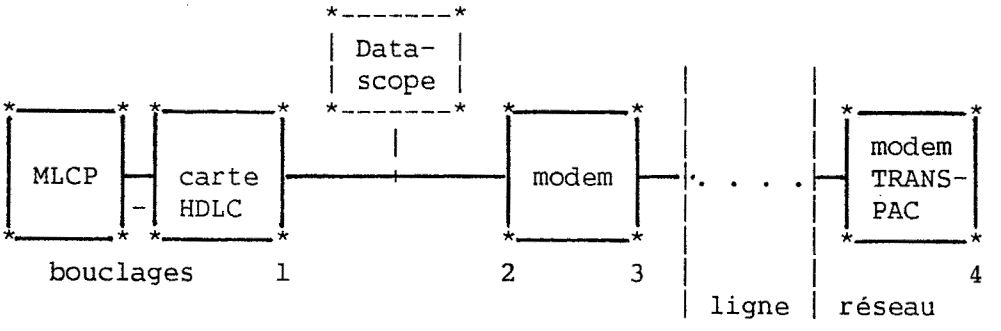


Figure 5.4 : outils de tests.

5.4.4.1. Le bouclage de ligne.

La configuration sur laquelle nous avons effectué les tests offre plusieurs possibilités :

- le bouclage 1 est positionné dans la carte HDLC par initialisation d'un indicateur dans une LCT. Ce type de test permet de vérifier le bon déroulement des procédures d'entrée-sortie mais ne fournit que peu d'indications sur la validité des résultats.

- le bouclage 2 se fait en entrée du modem local et n'a pas été utilisé.

- le bouclage 3 qui se fait en sortie du modem local est celui

qui a été le plus pratiqué car il est le plus complet étant donné que TRANSPAC n'autorise pas le bouclage 4 sur le réseau.

5.4.4.2. Utilisation du Datascope.

Le Datascope est un appareil qui, branché en série entre la sortie de la carte MLCP et le modem, nous permet de visualiser sur un écran tous les signaux qui passent sur la ligne avec certaines facilités comme la reconnaissance des fanions. Nous avons utilisé ce montage pour effectuer les premières vérifications du contenu des trames émises par le coupleur.

5.4.4.3. Les traces.

Nous avons mis en œuvre plusieurs types de traces :

- Pour vérifier le contenu de la mémoire RAM du coupleur après un test, nous avons utilisé un programme de vidage de la mémoire (mlcpdump) qui formate en LCT, CCP et CCB.

- Nous avons exploité les utilitaires Unix (hd et nm) pour obtenir le contenu des zones tampon et des variables du driver HDLC. L'utilitaire "nm" fournit les adresses des variables du noyau en mémoire centrale.

- Enfin le programme driver HDLC a été "tracé" avec une procédure rudimentaire de sortie de messages sur la console système en le compilant avec l'option "DEBUG".

Un problème particulier s'est posé pour la mise au point du module assembleur d'entrée-sortie (hdlc.s) qui nous a obligé à utiliser la trace système prévue pour le noyau (option TRACE).

Dans un premier temps, nous avons testé l'interface du niveau utilisateur avec un programme qui effectue des appels-système OPEN, READ, WRITE, IOCTL et CLOSE en bouclage de ligne. Ces tests nous ont permis d'obtenir un système qui répondait normalement dans ces conditions.

Lorsque le niveau supérieur d'X25 a été disponible, nous avons pu tester l'interface avec le driver "paquet". Cette phase a mis en évidence un certain nombre de problèmes au niveau HDLC

comme la perte d'interruptions évoquée en 5.4.3. Le système a ensuite été testé en exécutant un programme de copie de fichier par l'intermédiaire de TRANSPAC, ce qui a eu pour effet de charger de façon importante le niveau HDLC. Ce test nous a permis de constater que le coupleur n'est pas suffisamment rapide pour suivre la cadence en réception imposée par une ligne à 9600 bauds. Cela se traduit par une fréquence élevée d'overrun (environ 20%) sur les trames de données. Pour éliminer ce problème, il nous faut soit optimiser le code du programme CCP, soit réduire la vitesse de la ligne à 4800 bauds. Pour l'instant nous essayons de rendre le code plus performant.

Une des difficultés de la mise au point réside dans le fait que chaque modification dans le driver entraine la réfection complète du système (fichier /unix) et que les tests nécessitent un chargement (bootstrap) qui bloque les autres utilisateurs du mini-6.

5.5. Conclusion.

Les premiers tests effectués ont montré que l'implémentation du protocole HDLC dans un coupleur MLCP est réalisable et fonctionne. Cependant, il faut ajouter qu'une qualification complète du système ne pourra intervenir qu'après une période probatoire d'exploitation intensive qui mettra en évidence la réaction de l'ensemble face à des conditions réelles de traffic.

Nous pouvons simplement dire que nous avons montré la faisabilité des solutions préconisées, malgré les insuffisances du coupleur (manque de mémoire, faible jeu d'instructions, absence de ressource de temporisation), mais il nous reste à vérifier la capacité du système à absorber des conditions d'exploitation normales. Dans tous les cas, le savoir faire accumulé nous autorise à penser qu'avec un coupleur programmable mieux adapté, nous pourrons développer rapidement un système plus robuste. Le chapitre suivant évoque l'intégration du niveau supérieur d'X25 en généralisant sur les communications dans Unix.

6. Architecture générale du système.

6.1. Introduction

Jusqu'à présent, nous avons présenté la mise en œuvre de la couche liaison de données d'X25 avec le protocole HDLC. Nous allons maintenant aborder les choix d'architecture et d'implémentation retenus dans EIES pour les niveaux supérieurs. Nous étudierons ensuite une alternative possible avec la version 4.2 BSD d'UNIX développée à l'université de Berkeley et qui offre un mécanisme généralisé de communication entre processus.

6.2. Les choix d'architecture dans EIES

6.2.1. L'adressage.

Il existe plusieurs possibilités pour définir des adresses entre des objets, qu'ils se trouvent sur un site local ou distant. Nous pouvons prendre par exemple: ESP[84]

L'adressage à plat:

Ce type d'adressage ne nécessite pas de structure interne et il ne fait pas référence aux couches plus basses du système.

L'adressage hiérarchique:

Il comporte une structure interne dont une partie fait référence à la couche inférieure à laquelle il faut ajouter un suffixe. Elle définit des points d'accès aux services de la couche (N-1).

L'adressage partitionné:

Dans ce cas, l'adresse est construite à partir d'une liste des différents domaines adressables. Il ne fait pas référence aux couches inférieures.

Les traitements imposés par les types d'adressage à plat et partitionné sont plutôt complexes alors que l'adressage hiérarchique est traité de façon très simple : il suffit de re-

tirer le suffixe pour les communications sortantes et de le rajouter pour les communications entrantes. Cet avantage a déterminé le choix de l'adressage hiérarchique pour le modèle EIES en plus du fait qu'il se calque bien sur le principe d'architecture retenu.

Au niveau interface session, nous disposerons de trois éléments dans le champ adresse pour identifier:

- les points d'accès aux services session
- les points d'accès aux services transport
- les points d'accès aux services réseau.

La couche session quant à elle recevra les trois éléments suivants des niveaux supérieurs:

[a] l'utilisateur de la session

[b] le système pour un réseau local

[c] le site distant.

La couche transport (classe 3) reçoit les éléments [b] et [c] du niveau supérieur et transforme [b] en [d] qui est l'adresse réseau, grâce à la primitive "TRANSF1" et à la table "/etc/sites".

La couche réseau reçoit l'élément [d] et n'opère aucune transformation.

La couche transport classe 4 reçoit les éléments [b] et [c] et recherche l'adresse ethernet de l'entité passerelle grâce à la primitive "TRANSF2" et à la table "/etc/systems".

A l'extérieur des couches de communication, le système propose des outils de gestion des différantes tables d'adressage.

6.2.2. Choix pour les différentes couches

Nous avons vu dans le chapitre précédent que le niveau liaison de données d'X25 a été en grande partie installé dans le coupleur MLCP, il reste donc à faire un choix pour les couches supérieures. Compte tenu des liens étroits qui les unissent au matériel, les drivers des niveaux paquets X25 et ethernet sont

intallés dans le noyau UNIX. Cette solution présente les meilleures garanties de performances.

En ce qui concerne la couche du transport, il a été décidé de proposer deux types d'implémentations pour assurer une plus grande souplesse et permettre d'effectuer des comparaisons de performances:

[a] la première consiste à installer le service transport classes 3 et 4 dans le noyau UNIX.

[b] la seconde doit développer le service transport classe 4 dans un mode utilisateur, c'est à dire en dehors du noyau.

D'autrepart, il a paru intéressant de donner à certains processus application la possibilité d'accéder directement au driver X25 aussi bien qu'au service transport. Cette facilité est intégrée par le biais de l'interface standard des appels systèmes comme le montre la figure 6.1.

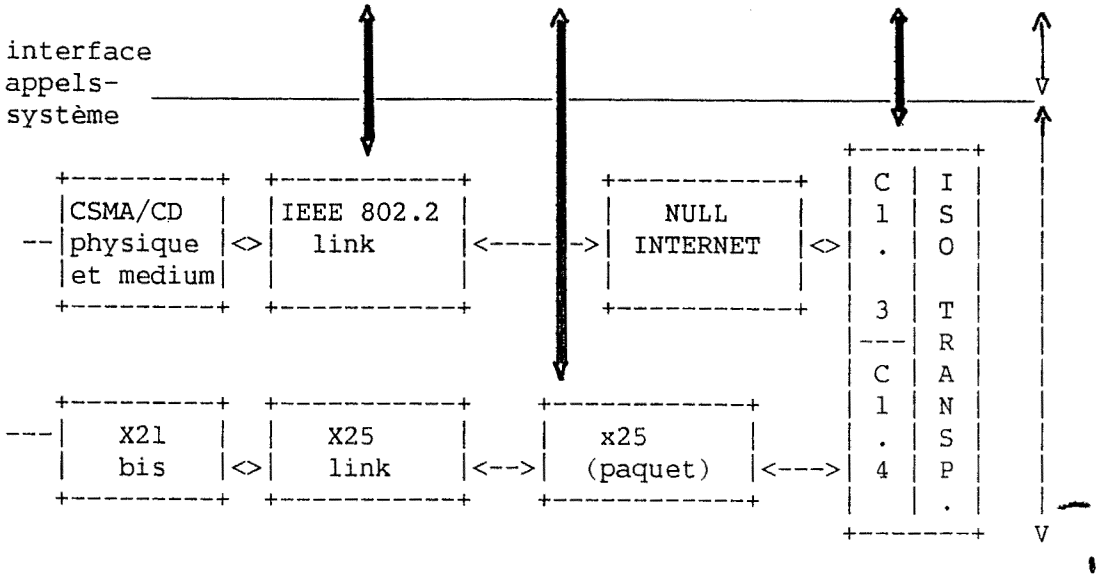


Figure 6.1: Services EIES dans le noyau UNIX

Pour conserver une approche homogène dans tous les problèmes d'échange au niveau utilisateur, il a été nécessaire de développer un mécanisme de communication entre processus (IPC) puis de généraliser ce concept dans le driver EIES.

Enfin un système de gestion de mémoire partagée vient complèter l'ensemble.

6.2.3. Le pseudo-périphérique IPC.

Ce mécanisme doit permettre de faire communiquer dans les deux directions des processus qui n'ont pas de liens de parenté (ce qui exclut les pipes) sans modifier le noyau. Pour respecter la philosophie générale d'UNIX, il a donc été conçu sous la forme d'un nouveau driver.

Ce driver doit pouvoir offrir des moyens de contrôler les communications comme:

- les attentes multiples
- les écritures et lectures conditionnelles (non bloquantes)
- un contrôle de flot.

L'IPC utilise le concept de port qui est représenté par un nom pris dans le contexte UNIX et associé à une mémoire tampon de stockage. Le port requière une synchronisation du type producteur-consommateur pour résoudre le problème des ressources qui sont allouées à la création.

6.2.4. Le driver EIES.

Ce driver doit essentiellement répondre à deux fonctions:

- La première consiste à fournir un moyen de représenter les relations de type client-serveur entre les entités des couches (N) et (N-1) de l'architecture.
- La seconde donne la possibilité de construire un mécanisme d'interaction non bloquant entre une tâche du niveau utilisateur et un serveur situé dans le noyau.

Une entité du niveau utilisateur (au sens donné par UNIX) ne pourra donc communiquer avec un serveur que par l'intermédiaire du driver EIES, ce qui présente une bonne garantie d'homogénéité et de sécurité.

6.2.5. La gestion de mémoire partagée.

Ce mécanisme est proposé sous la forme d'un pseudo-périphérique avec son driver associé pour éviter les recopies inutiles de données au cours des communications entre les différentes couches du logiciel. Son principe est simple: il consiste à copier une première fois les données dans une chaine de mémoire réservée puis à les manipuler ensuite par l'intermédiaire d'un identificateur de chaine et de primitives appropriées.

6.2.6. Schéma général de l'implémentation.

A ce niveau de la présentation, c'est le dessin de la figure 6.2 qui nous permettra de mieux saisir l'ensemble des mécanismes mis en œuvre pour réaliser le système de communication EIES.

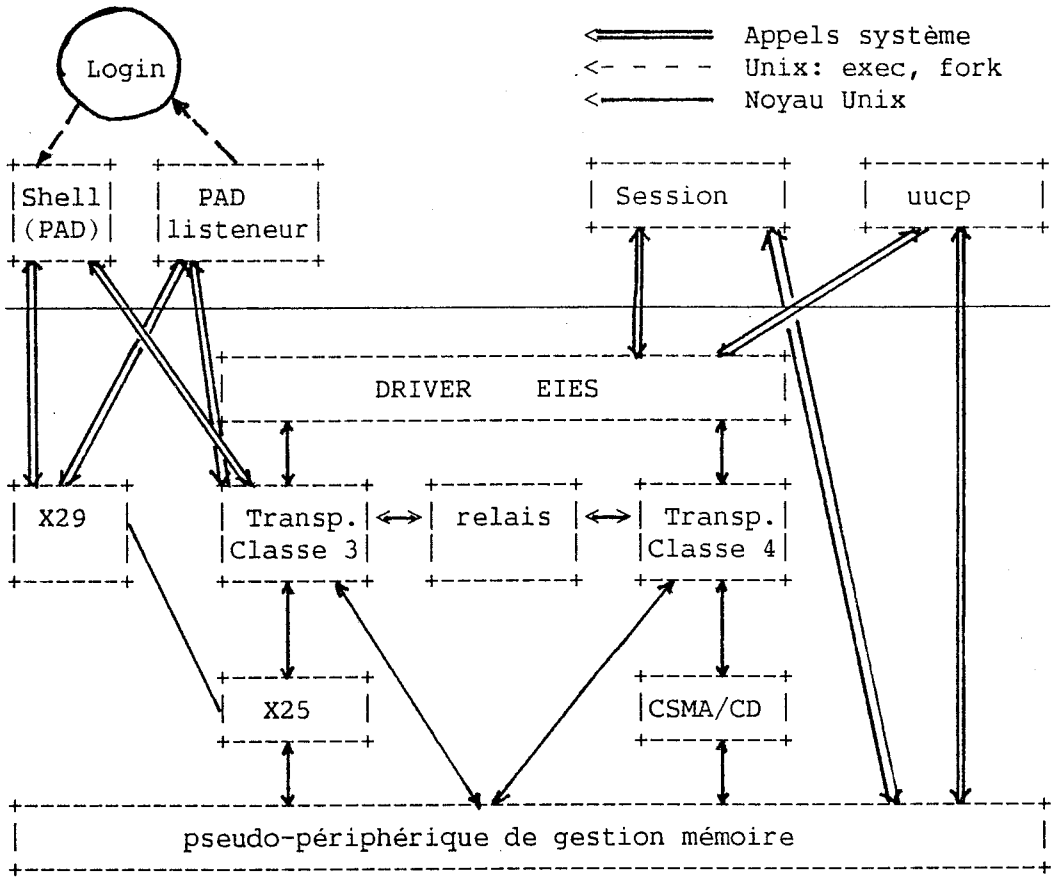


Figure 6.2: Architecture générale du système EIES.

6.2.7. Spécification du niveau paquet X25.

Dans EIES, le service de circuit virtuel commuté est géré par deux pseudo-drivers XNC et SVC:

- "XNC" qui supporte les appels système open(), close() et ioctl() permet de contrôler la ligne HDLC utilisée pour l'établissement des circuits virtuels.

- "SVC" contient tout le protocole du niveau paquet tel qu'il est spécifié dans la norme X25. La mise en œuvre de l'interface utilisateur est fait par l'intermédiaire des appels système open(), close(), read(), write() et ioctl().

Un processus serveur est chargé de gérer les demandes de connexion et la phase de transfert d'informations autorise l'emploi d'options telles que:

- lecture non bloquante
- lecture et écriture en utilisant la gestion de mémoire partagée
- le mode de travail avec le mode "raw" qui n'opére aucune interprêtation sur les caractères et le mode "cooked" qui donne une certaine visibilité du terminal.

6.3. Un mécanisme généralisé de communication entre processus.

La présentation générale du modèle EIES a bien mis en évidence la nécessité d'améliorer les communications entre processus dans UNIX, c'est pourquoi nous présentons dans la section suivante une réponse originale à ce problème avec la version 4.2bsd d'UNIX. Nous essayerons ensuite de vérifier si le modèle EIES peut être implémenté facilement et avec efficacité sur cette version pour bénéficier de tous les avantages qu'elle apporte.

6.3.1. Objectifs de la version UNIX 4.2bsd.

D'après ses auteurs (4.2bsd[a,b,c,d,e]) la version 4.2 d'UNIX est le résultat d'une évolution constante de la version de base et elle apporte de nombreuses améliorations concernant les performances, la gestion de fichiers, la gestion mémoire et surtout les communications.

Dans le domaine des communications, l'objectif principal est de construire un sous sytème intégrant un mécanisme généralisé capable de fournir à l'utilisateur les moyens de réaliser une application répartie. L'originalité du système consiste à proposer des outils plutôt que des solutions complètes et obligatoirement figées en réponse aux différents problèmes.

En résumé, le sous-système de communication doit fournir un moyen simple et efficace pour transférer des données entre des processus indépendants et autoriser un accès cohérant à des ressources distribuées (serveurs).

Certains objectifs ont été délibérément écartés, par exemple la protection et le contrôle d'accès restent au niveau local et la transmission d'information n'est pas structurée.

6.3.2. Le modèle de base.

Le modèle du sous-système de communication repose sur deux concepts qui sont les DOMAINES et les SOCKETS

Le DOMAINE définit un environnement de communications avec un espace d'adressage et des noms qui lui sont associés. La version courante de la 4.2bsd connaît deux domaines qui sont le domaine UNIX (machine locale) et le domaine INET (interconnexion de réseau).

Le SOCKET est une abstraction qui représente la destination de toute communication à l'intérieur d'un domaine. Il faut "lier" un nom à un socket pour pouvoir le référencer et chaque socket possède un TYPE ainsi qu'un ou plusieurs processus associés.

Trois TYPES sont actuellement proposés:

STREAM: qui offre un service de circuit virtuel avec transfert bidirectionnel et garantie de séquencement sans duplications ni pertes.

DGRAM : qui offre un service de datagrammes bidirectionnel mais sans garantie de séquence, d'arrivée ou de duplication.

RAW : qui est utilisé par les protocoles de communication des couches basses du système.

6.3.3. L'interface utilisateur.

L'interface utilisateur repose sur de nouveaux appels systèmes que nous allons évoquer.

6.3.3.1. Création d'un socket.

Un nouveau socket peut être crée par l'appel système

```
s = socket(domaine,type,protocole);
```

qui permet de préciser le domaine de communication, le type de services requis et le protocole utilisé.

6.3.3.2. Association d'une référence.

Pour pouvoir être employé, un socket doit avoir un nom de référence qui sera affecté par l'appel système

```
bind(s,nom,longueur du nom);
```

"s" est un descripteur retourné par l'appel socket. Le nom sera interprêté par le protocole en fonction du domaine de communication.

6.3.3.3. Etablissement d'une connexion.

Il existe une possibilité de connexion avec un socket "serveur" avec l'appel système

```
connect(s,nom du serveur,long. du nom);
```

Pour que le "rendez-vous" ait lieu, il faut que le serveur effectue deux opérations:

[a] il écoute les demandes de connexion par :
```
listen(s,nb);
```
nb étant le nombre maximum de demandes en attente.

[b] il doit accepter la connexion par :
```
descripteur = accept(s,nom du client, long.du nom);
```
cet appel est normalement bloquant.

6.3.3.4. transfert des données.

Après la connexion du socket, l'échange d'informations devient possible avec les appels standards de read() et write() ou avec les nouveaux appels send() et recv() qui comportent des options comme les données express ou la prélecture.

Notons qu'il existe une possibilité d'échange avec des sockets sans connexion avec les appels sendto() et recvfrom(). Cette facilité est réservée au datagrammes.

6.3.3.5. Destruction des sockets.

la destruction d'un socket s'obtient grâce à l'appel close() qui préserve les données en cours de transfert. Si le close() est précédé de shutdown(), alors les données en cours seront perdues.

6.3.3.6. Multiplexage des entrées-sorties.

Le multiplexage des entrées-sorties peut se faire de façon synchrone avec l'appel système du type select d'ADA (ADA[84]):

```
select(nb.desc,desc.read,desc.write,desc.execp,timeout);
```

Cette fonction retourne les descripteurs des sockets pour lesquels il existe des données à lire ou à écrire sous la forme de masques binaires.

6.3.4. Les fonctions réseau.

La version 4.2bsd d'UNIX offre une série de fonctions qui permettent de définir et de manipuler les adresses. Les objets référencés sont les systèmes hôtes, les différents réseaux, les protocoles utilisés et les services disponibles.

Une structure particulière est associée à chaque type d'objet, et le regroupement de ces structures dans des fichiers comme "/etc/hosts" ou "/etc/services" constitue une base de données de tous les éléments adressables du système de communication.

Les fonctions réseau sont développées avec un soucis de flexibilité qui doit permettre de conserver un interface utilisateur commun lorsque de nouveaux protocoles sont pris en compte. Les modules standards proposés assurent les transformations suivantes sur les adresses:

- nom d'hôtes en adresse réseau
- nom de réseaux en numéro de réseau
- nom de protocoles en numéro de protocole
- nom de services en numéro de port et de protocole à employer dans les communications avec le processus serveur.

Parmis les modules de gestion du réseau, nous ne donnerons en exemple que ceux concernant les systèmes hôtes, les autres étant construits suivant le même principe.

gethostbyname utilise le nom d'hôte pour retrouver la structure correspondante dans le fichier "/etc/hosts".

gethostbyaddr retourne la structure du premier système hote rencontré dont l'adresse correspond à celle qui lui est indiquée.

gethostent délivre la structure suivante dans la base de données.

La combinaison de ces différentes fonctions doit pouvoir résoudre les problèmes d'adressage qui se posent à l'utilisateur.

6.4. Transposition du modèle EIES sur la 4.2bsd.

6.4.1. Objectif.

Le principal objectif poursuivi avec la transposition des développements effectués pour EIES consiste à étendre le champ d'utilisation de ce système tout en bénéficiant des nombreux atouts de cette version comme:

- les performances améliorées

- les nouveaux outils dont elle dispose
- un bon environnement de communication
- son utilisation très répendue dans les centres de recherche.

6.4.2. Comparaison des deux architectures.

6.4.2.1. Adressage.

A ce niveau, le modèle EIES fait un choix avec l'adressage hiérarchique (6.2.1) tandis que la version 4.2bsd laisse une liberté d'action plus grande en proposant plusieurs schémas d'adressage généralement liés à un domaine de communication.

6.4.2.2. Modèle de base.

Le modèle EIES s'appuie principalement sur l'ouverture du système UNIX qui autorise la création facile de nouveaux périphériques ou pseudo-périphériques en ajoutant des "drivers". Ce choix apporte une grande modularité dans le développement et il correspond bien à la philosophie UNIX sur les entrées-sorties. De plus il offre une couche de logiciel commune pour l'interface utilisateur avec les appels-système.

C'est ainsi que nous avons prévu de construire des drivers (ligne HDLC, paquet X25 et transport) avec pour chacun d'eux:

- un protocole programmé
- un interface utilisateur
- un interface noyau ou interne.

A ces éléments de base, il faut ajouter des outils communs avec les drivers EIES et mémoire partagée.

La philosophie retenue pour la version 4.2bsd de Berkeley est très différente puisqu'elle conduit à proposer un sous-système de communication pratiquement indépendant. Ce sous-système ne prétend pas répondre directement à un problème parti-

culier comme celui posé dans les objectifs d'EIES, mais il met à disposition une panoplie d'outils et de mécanismes qui doivent permettre de le faire simplement.

Des concepts comme les domaines de communications ou les sockets offrent des facilités intéressantes:

- la programmation de protocoles indépendants des services
- un interface utilisateur standard et puissant avec les nouveaux appels systèmes sur les sockets
- la possibilité de construire rapidement des processus serveurs.

Les deux architectures présentent des similitudes avec la gestion de mémoire partagée, le concept d'IPC ou la définition de passerelles vers des processus serveurs (les "PORTS" dans EIES ou "PORTALS" dans 4.2bsd).

6.4.3. Les problèmes d'implémentation.

Avec la version 4.2bsd, la prise en compte de nouveaux protocoles est facilitée grâce à l'existence de modules utilitaires qui permettent de manipuler les nouvelles structures de données. Ces données décrivent la représentation interne des concepts mis en œuvre dans le sous-système de communication. C'est ainsi que nous trouverons des structures pour représenter:

Les SOCKETS
: Cette structure comprend une chaine de "buffers" de réception et d'émission ainsi qu'un pointeur sur les modules chargés de supporter les services.

Les PILES de DONNEES
: pour la gestion des espaces mémoire alloués aux sockets.

Les PROTOCOLES
: Cette structure regroupe des pointeurs sur différents modules de gestion et interfaces.

Les TABLES de ROUTAGE

La figure 6.3 nous montre que le noyau du système comprend trois couches de logiciel qui définissent trois types d'interfaces.

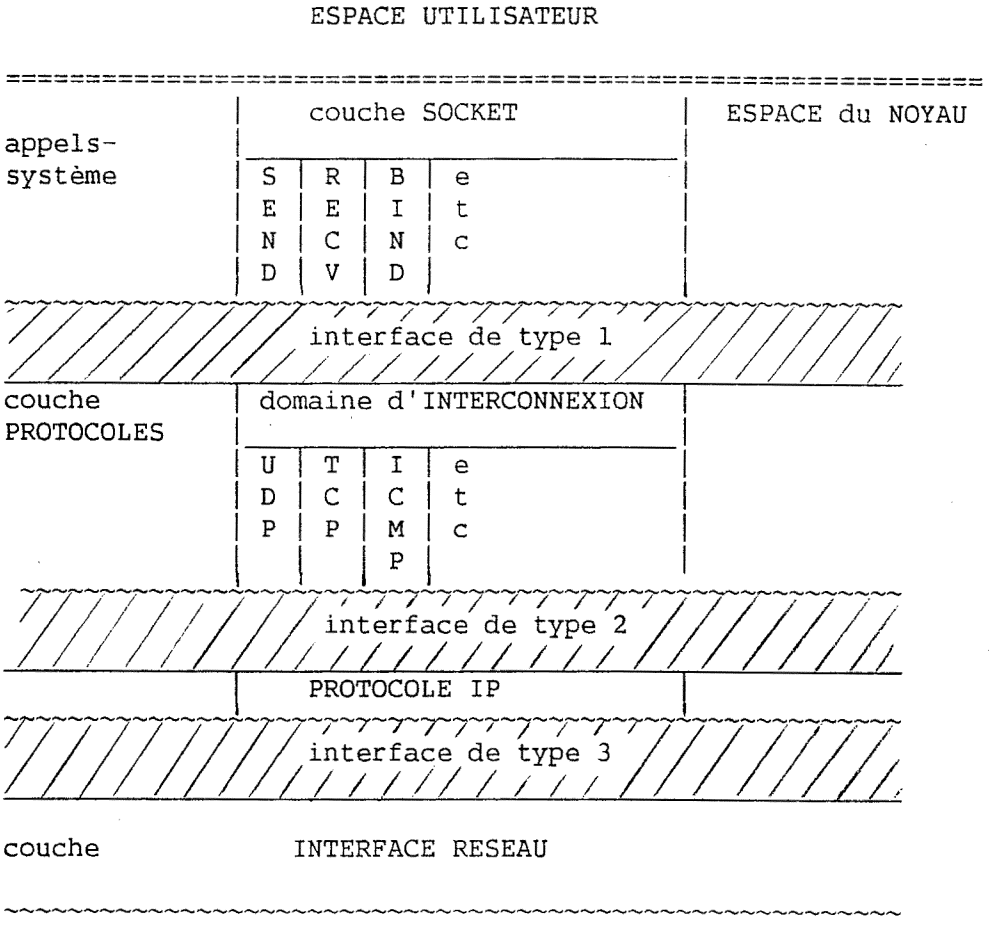


Figure 6.3: Couches internes du sous-système de communication

Les concepteurs du système ont prévu de standardiser les différents types d'interface en proposant des fonctions pour chacun d'eux.

Au niveau de l'interface SOCKET/PROTOCOLE (type 1), nous trouvons le module:

pr_usrreq()

qui supporte un grand nombre de requêtes du type:

ATTACH qui appelle le protocole nécessaire à une liaison

LISTEN qui indique que l'utilisateur souhaite prendre en compte les appels entrants

RCVD signale que l'utilisateur a lu les données qui lui étaient destinées

etc...

L'interface entre les différentes couches de protocole (type 2) est réalisé par l'intermédiaire de deux classes de fonctions:

- Celles qui traitent les transferts de données avec:

  pr_output() qui gère les transferts à destination du réseau.

  pr_input() qui traite les flux dirigés vers l'utilisateur.

- Celles qui qui sont chargées de faire passer de l'information de contrôle à l'utilisateur:

  pr_ctlinput()

  pr_ctloutput()

Notons que ces modules sont très dépendant du protocole utilisé.

Le dernier type d'interface entre les PROTOCOLES et l'INTERFACE-RESEAU (type 3) se développe en deux temps:

- Pour la transmission de paquets, il faut utiliser la fonction:

  if_output()

  qui provoque la mise en file d'attente d'émission du paquet et lance le module de transfert de données qui appartient au niveau interruption.

- La réception d'un paquet se produit dans la file d'attente de réception qui est associée à chacun des PROTOCOLES de bas niveau et elle déclenche l'envoi d'un

signal qui doit initialiser le traitement de lecture du paquet. A ce stade des macro-intructions sont prévues pour empiler ou dépiler les paquets (IF_ENQUEUE, IF_DEQUEUE et IF_PREPEND).

Enfin une dernière facilité est offerte avec les SOCKETS "RAW" qui autorisent un accès direct de l'utilisateur au protocoles de bas niveau. Cette caractéristique donne la possibilité de développer de nouveaux PROTOCOLES dans l'espace utilisateur tout en bénéficiant des services proposés par les PROTOCOLES de bas niveau.

D'aprés les éléments que nous venons de voir, l'installation des services prévus dans le système EIES pourra se faire dans la version 4.2bsd en isolant les PROTOCOLES des niveaux TRANSPORT et X25 afin de les standardiser et de les reporter dans la couche interne PROTOCOLE du sous-système de communication. Il sera également nécessaire de développer du code à l'intérieur des différents modules d'interface que nous avons mentionnés ci-dessus.

6.5. Conclusion.

Dans ce chapitre, nous avons examiné deux réponses possibles au problème de l'interconnexion de systèmes exploités sous UNIX. Dans un premier temps, la solution apportée par la version 4.2bsd peut paraitre plus séduisante car elle met en œuvre un sous-système généralisé qui semble pouvoir s'adapter de manière élégante à un grand nombre de situations. Par contre, ce sous-ensemble est spécifique à la version d'UNIX développée à l'Université de Berkeley et sous cet aspect, il ne répond pas aux contraintes fixées dès le départ du projet EIES. En effet, le critère de portabilité a toujours prévalu dans les choix effectués et c'est la version "V7" d'UNIX qui a été considérée comme standard en la matière.

Le modèle d'architecture retenu dans EIES est donc justifié dans la mesure où les modifications apportées pour remplir les fonctions définies dans le projet sont très modulaires et facilement portables d'un système à un autre sans remise en cause de l'existant.

Bien que les deux solutions envisagées, EIES et 4.2bsd, semblent bien adaptées chacune à leur contexte, la section 6.4.2 nous montre qu'il serait intéressant de profiter de leur complémentarité en intégrant les protocoles développés pour EIES

dans la version UNIX 4.2bsd. Cette ouverture du modèle EIES permettrait d'étendre considérablement le champ d'utilisation du système qui constituerait ainsi un outil très efficace au service des chercheurs de la communauté Européenne dans le cadre du projet ESPRIT.

7. Conclusions générales.

Dans ce mémoire, nous avons pu constater au chapitre 1 que les hauts responsables de la communauté européenne ont bien pris concience de l'urgence de participer à la compétition internationale pour la maîtrise des techniques de traitement de l'information. Avec le programme ESPRIT, nous pensons qu'ils ont su s'en donner les moyens.

Parmi ces techniques, nous avons particulièrement étudié celles qui se rapportent aux problèmes de communications à travers notre participation au développement du système d'échange d'informations EIES. Ce travail a mis en évidence la complexité des problèmes d' interconnexion de réseaux hétérogènes et les progrès accomplis dans leur compréhension grâce à la normalisation et au modèle de référence proposé par l'ISO.

La réalisation des couches de protocole de la norme X25 du CCITT sur le mini-6 nous a permis de mieux appréhender les mécanismes mis en œuvre dans un protocole comme HDLC et de concrétiser les notions d' interface , de services ou de point d'accès. Cela nous a également permis de bien maitriser les fonctions d'un système d'exploitation , en l'occurrence UNIX, avec en particulier le fonctionnement des entrée-sorties et la communication entre processus.

Nous avons souligné les progrès technologiques constants qui sont réalisés d'année en année ainsi que l' accroissement des besoins en matière de communication, que ce soit sur le plan des performances ou sur celui de la fiabilité. C'est pourquoi nous pensons que le travail réalisé dans le projet EIES devrait logiquement se poursuivre avec la prise en compte des nouvelles techniques présentes sur le marché ou à venir, comme les liaisons par fibres optiques ou par satellite pour offrir de nouveaux services. Ces services pourraient s'orienter vers la transmission de l'image ou de la parole qui nécéssitent des débits très importants.

L'augmentation des débits passera obligatoirement par une intégration de plus en plus importante des composantes logicielles sur des circuits VLSI très performants. Nous rejoignons ainsi un autre domaine qui est lui-même exploré par le programme de recherche ESPRIT. Cela montre bien l'imbrication des différentes techniques et l'importance d'être présent dans tous les secteurs concernés par le traitement de l'information.

Nous conclurons en souhaitant que le programme ESPRIT atteigne ses objectifs, que nous pensons bien adaptés à la situation actuelle, pour donner un nouveau souffle salutaire à toute l'industrie Européenne.

- 30 août 1984 -

Bibliographie

ADA[84] Reference manuel for the Ada programming language
ALSYS ANSI/MIL-STD1815 A

AFNOR[82] Modèle de référence de base pour l'interconnexion de systèmes ouverts
AFNOR Z70-001 Juin 82

BOUR[82] The Unix system - S.R. BOURNE
Addison-Wesley publishing company 1982

CII[80] Minicomputer systems handbook
CII-HONEYWELL BULL Documentation center Dec 1980

CII[81] Mini-6 communications handbook
CII-HONEYWELL BULL Sep 1981

COM[84] Commission of the European Communities
draft council decision 56 final february 1984

CORN[81] Systèmes informatiques répartis - CORNAFION
Dunod informatique 1981

ESP[84] Esprit Information Exchange System
Technical specifications april 1984

KAAR[83] Unix operating system - C. KAARE
J. Wiley interscience 1983

KERN[81] The C programming language - B.W. KERNIGHAN, D.M. RITCHIE
Prenctice-hall inc New Jersey

KESS[80] Use of the HDLC protocol in process control systems with multi-primary communication - H.KESSMAN
Hahn-Meitner Institut Berlin avril 1980

MACC[81] Téléinformatique - C.MACCHI,J F GUILBERT
Dunod informatique 1981

TRAN[83] Spécifications techniques d'utilisation du réseau TRANSPAC
Télécommunications 1983

UBI[83] UBIES final report - BULL, GEC, ICL, SIEMENS
1983

WEISS[83] Bit oriented data link controls - A.J. WEISSBERGER
Computer design march 1983

X25[84] Avis X25
fascicule VIII.2 p100-189 1984

4.2BSD[a] An architecture for Interprocess Communication in Unix
JOY FABRY 1981

4.2BSD[b] Proposals for enhancement of Unix on the VAX
JOY FABRY 1981

4.2BSD[c] 4.2bsd System manuel
JOY COOPER FABRY LEFFLER KUSICK 1982

4.2BSD[d] A 4.2bsd Networking Implementation Notes
LEFFLER FABRY JOY 1983

4.2BSD[e] A 4.2bsd Interprocess Communication - Primer draft
LEFFLER FABRY JOY 1983

TABLE DES MATIERES

TYPE DE TRAME	CATEGORIE	CODAGE 8 7 6	5	4 3 2	1
1. Information:					
I - Données	Commande	N(R)	P	N(S)	0
2. Supervision:					
RR - prêt à recevoir	Réponse / Commande	N(R)	P/F	0 0 0	1
REJ - Rejet	Réponse / Commande	N(R)	P/F	1 0 0	1
RNR - Non prêt à recevoir	Réponse / Commande	N(R)	P/F	0 1 0	1
3. Non numérotées:					
DISC - Déconnexion	Commande	0 1 0	P	0 0 1	1
UA - Accusé de réception	Réponse	0 1 1	F	1 1 1	1
CMDR - Rejet commande	Réponse	1 0 0	F	0 1 1	1
FRMR - Rejet de trame					
SABM - Connexion ou Réinitialisation	Commande	0 0 1	P	1 1 1	1
DM - Indication de mode déconnecté	Réponse	0 0 0	F	1 1 1	1

REMARQUE: nous ne considérons que la procédure LAPB.

Répertoire des Commandes et Réponses HDLC.

NOM :

hdlc (4) - interface général pour les lignes hdlc.

DESCRIPTION :

Le driver HDLC fournit un moyen de communiquer par l'intermédiaire d'une ligne asynchrone. Il permet une connexion bi-directionnelle simultanée entre des processus utilisateurs.

Les fichiers spéciaux "/dev/hdlc*" font référence à des lignes physiques.

Les processus utilisateurs peuvent accéder aux lignes hdlc au moyen des appels système open, close, read, write et ioctl (2).

Lors de sa première ouverture, une ligne hdlc est rattachée à un utilisateur, ce qui a pour effet de démarrer le protocole de niveau 2 d'x25 et de réserver des buffers d'entrée-sortie.
En cas d'erreur, open (2) retourne la valeur (-1) et la variable ERRNO précise le type d'erreur:

ENXIO : la ligne ne répond pas

EBUSY : la ligne est déja réservée

Les accès à une ligne hdlc ne sont pas exclusifs et plusieurs processus peuvent ouvrir une ligne simultanément.

Les appels système read et write (2) sont utilisés pour recevoir et émettre des données sur la ligne.
Ces appels sont bloquants et les processus utilisateurs sont "endormis" si aucune donnée n'est disponible en réception ou si les buffers d'émission sont tous occupés.
Read et write (2) retournent la valeur (-1) en cas d'erreur ou bien le nombre de caractères effectivement transmis.
La variable ERRNO peut contenir les codes :

ENXIO : la ligne ne répond pas

L'appel système ioctl (2) ne supporte actuellement qu'une seule fonction qui fournit des données statistiques sur l'exploitation de la ligne:

ioctl(fildes, cmd, arg);

cmd = CLSTAT
arg = pointeur sur une structure xncstat qui est définie dans <x25.h>.

L'appel système close (2) opère la déconnexion de la ligne, ce qui a pour effet de:

- suspendre le protocole X25 niveau 2
- libérer les buffers d'entrée-sorties
- annuler la réservation de la ligne.

Le close retourne la valeur (-1) en cas d'erreur avec un code erreur:

ENXIO = la ligne n'est pas connectée ou des informations sont encore en cours de transit.

Certains paramètres de la ligne sont prédéfinis dans le driver hdlc et avec le niveau actuel d'implémentation, il n'est pas possible de les modifier dynamiquement. Il s'agit de:

WINDOW	7	nombre de trames émises en anticipation
WATCHDOG	8	valeur du temporisateur en secondes
LGBUF	132	longueur maxi d'un paquet en caractères
MAXRETRANS	10	nombre maxi de retransmission pour un même paquet
RCVBUF	8	nombre de buffers de réception
SENDBUF	8	nombre de buffers d'émission

FICHIERS:

"/dev/hdlc*"

~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~

1. CORPS DU PROGRAMME RECEPTION

```
        /* boucle sur détection de début de trame */

Initialisations
TANTQUE la ligne est active
   SI début de trame APPEL "TRAME"
   FSI
FTANTQUE
```

1.1. TRAME

```
            /* début de réception d'une trame */

Initialisations (statut, code erreur, etc)
lecture du 1er caractère et stockage dans adresse reçue
SI fin de trame APPEL "TRAME-2"
SINON           APPEL "TRAME->2"
FSI
initialition du FCS dans LCT3-4
```

1.1.1. TRAME-2

```
              /* trame de longueur 2 octets */

lecture du 2eme caractère et stockage dans commande reçue
SI trame différent de (info ou FRMR) APPEL "SUPERV"
FSI
```

1.1.2. TRAME->2

```
        /* trame de longueur supérieure à 2 octets */

lecture du 2eme caractère et stockage dans commande reçue
SELON commande reçue :
   QUAND trame information APPEL "INFORM"
   QUAND trame (FRMR) APPEL "FRMR"
FSELON
```

1.1.2.1. SUPERV

```
   /* trame supervision ou non numérotée sauf (FRMR) */

APPEL "DECOD"
APPEL "CTRLFCS"
SELON code erreur :
   QUAND code erreur 01 EXIT "SUPERV"  /* Abandon de la  trame
   */
   QUAND code erreur 02 APPEL "REINIT"; EXIT "SUPERV"
FSELON
APPEL "ACK"  /* Procédure d'aquittement */
APPEL "HDLC"  /* Procédure traitement suivant  commande  reçue
*/
```

1.1.2.2. FRMR

```
                    /* trame (FRMR) */

APPEL "DECOD"
SI code erreur non 00 EXIT "SUPERV" /* Abandon de la trame */
TANTQUE non fin de trame
   lecture et abandon caractères info
FTANTQUE
APPEL "CTRLFCS"
SI code erreur 00   /* traitement trame (FRMR) */
   SI phase déconnexion indication de déconnexion
   SINON APPEL "REINIT"
   FSI
FSI
```

1.1.2.3. INFORM

```
                /* trame d'information */

APPEL "DECOD"
SELON code erreur :
   QUAND code erreur 01 EXIT "INFORM"
   QUAND code erreur 02 APPEL "REINIT"; EXIT "INFORM"
   QUAND code erreur 03 APPEL "ACK"; APPEL "REJET"; EXIT  "IN-
   FORM"
FSELON
TANTQUE non fin de trame
   lecture caractère info
   SI bloc valide stockage dans CDB
   SINON          erreur overflow (04)
```

```
      FSI
   FTANTQUE
   SI erreur synchro, abort ou overrun : code erreur = 04
   FSI
   APPEL "CTRLFCS"
   SI code erreur 04 /* indication trame I erronée au niveau
   supérieur */
      incrémentation compteur gnb
      interruption mémoire centrale
      passage au CCB suivant (gnb)
   SINON APPEL "ACK", /* traitement trame (I) */
      incrémentation V(R) modulo 8
      reset état REJET
      demande émission (RR)
      incrémentation compteur gnb
      interruption mémoire centrale
      passage au CCB suivant
   FSI
```

CONTENU DES FONCTIONS APPELEES

```
DECOD   /* Décodage des champs Adresse et Commande */
   retour erreur 01 = adresse erronée ou bit F invalide
   retour erreur 02 = champ N(R) pas dans intervalle {DN(R),V(S)}
   retour erreur 03 = champ N(S) différent de V(R)

ACK     /* Pour trames (I, RR et REJ), acquittement des trames
        reçues jusqu'à N(R)-1 */
   SI N(R) = V(S) : indication désarmement timer
   SINON          indication armement timer
   FSI
   SI trame (RR ou REJ) interruption mémoire centrale
   FSI

HDLC    /* Traitement des trames suivant la valeur du champ de
        commande */
   SELON champ commande reçue :
      QUAND (REJ)
         SI (F=1) reset état REVEIL
         FSI
         SI N(R) # V(S)
            V(S) := N(R)
            état ligne = réémission
            indication de REJET
            interruption mémoire centrale
         FSI
```

```
   QUAND (RR)
      SI (état REVEIL et F=1) APPEL "REJ"
      FSI
   QUAND (DISC)
      SI phase échange information demande émission (UA)
      FSI
      état ligne := fermée
      indication de déconnexion
      interruption mémoire centrale
   QUAND (UA)
      SI phase ouverture ou réinitialisation
         état ligne := établi
         indication de connexion
      FSI
      SI phase de déconnexion
         état ligne := fermée
         indication de déconnexion
      FSI
      interruption mémoire centrale
   QUAND (DM)
      SI phase de fermeture
         état ligne := fermée
         indication de déconnexion
         interruption mémoire centrale
      SINON APPEL "REINIT"
      FSI
   QUAND (SABM)
      SI phase ouverture
         état ligne := établi
         indication de connexion
      SINON
         demande émission (UA)
         état ligne := réinitialisation
         indication de connexion
      FSI
      interruption mémoire centrale
FSELON

REINIT  /* demande de réinitialisation de la ligne  au  niveau
        supérieur */
   reset état ligne
   indication de réinitalisation (INTER)
   interruption mémoire centrale

REJET   Demande d'émission d'une trame (REJ) et changement  du
        code état ligne
```

2. CORPS DU PROGRAMME EMISSION

2.1. CONNEXION

```
                    /* phase de connexion */

R-à-z des variables ligne (V(S), V(R), DN(R) ...)
Initialisation état ligne et champ adresse émission
Chargement du champ commande avec (SABM,P=1)
APPEL "SENDFR" /* émission de la trame */
Mise en attente
```

2.2. DECONNEXION

```
                    /* phase de déconnexion */

Initialisation état ligne et champ adresse émission
Chargement du champ commande avec (DISC,P=1)
APPEL "SENDFR" /* émission de la trame */
Mise en attente
```

2.3. ETABLI

```
                /* phase d'échange d'information  */

TANTQUE la ligne est active
Initialisation champs Adresse et Commande
SI demande émission trame (UA) APPEL "SDTUA"; EXIT "ETABLI"
FSI
SELON code état ligne :
   QUAND reset ALLER_A "CONNEXION"
   QUAND établi
      SI demande réémission (code état)
         reset code état
         reset mot d'interruption
         mise en attente.
      FSI
      SI expiration timer (mot d'interruption)
         reset mot d'interruption
         SI DN(R) différent de  V(S)  APPEL  "TIMER";  EXIT
         "ETABLI"
         FSI
      FSI
      SELON demande d'émission (mot SENDT)
```

```
            QUAND (REJET avec P=1) APPEL "SDREJ1"; EXIT "ETA-
            BLI"
            QUAND (REJET avec P=0) APPEL "SDREJ0"; EXIT "ETA-
            BLI"
            QUAND (RR avec F=1) APPEL "SDRR1"; EXIT "ETABLI"
         FSELON
         SI état ligne REVEIL
            SI demande émission (RR) APPEL "SDRR00"; EXIT
            "ETABLI"
            SINON EXIT "ETABLI"
            FSI
         FSI
         SI il exite un CCB valide
            SI demande émission (RR) APPEL "SDRR0"; EXIT "ETA-
            BLI"
            SINON EXIT "ETABLI"
            FSI
         FSI
         SI V(S) est dans la fenêtre d'émission APPEL "SDTRI";
         EXIT "ETABLI"
         SINON
            SI demande d'émission (RR) APPEL "SDRR0"; EXIT
            "ETABLI"
            FSI
         FSI
      FSELON
   FTANTQUE
```

CONTENU DES FONCTIONS APPELEES

```
SDTUA    reset mot SENDT
         SI état ligne RESET initialisation des variables ligne
         FSI
         chargement de la commande avec (UA, F=1)
         APPEL "SENDFR"

SDREJ1   chargement de la commande avec (F=1)
         APPEL "SDREJ0"

SDREJ0   Chargement commande avec (REJ)
         Mise en forme de N(R) dans le champ commande
         APPEL "SENDFR"
         reset RR et REJ

SDRR1    chargement de la commande avec (F=1)
         APPEL "SDRR0"
```

SDRR0 Chargement commande avec (RR)
Mise en forme de N(R) dans le champ commande
APPEL "SENDFR"
reset RR

TIMER Chargement code état = REVEIL
Chargement mot d'interruption = REEX
Chargement mot WT2 = DN(R)
interruption mémoire centrale
mise en attente

STIMER chargement de la commande avec (P=1)
Chargement mot d'interruption = armement timer
APPEL "FINTRI"

SDTRI SI V(S) = DN(R) Chargement mot d'interruption = armement timer
FSI
calculer V(S) := V(S)+1 modulo 8
APPEL "FINTI"

FINTI initialisation champ adresse
mise en forme de V(S) et V(R) dans champ commande
APPEL "SENDFR"
reset RR

SENDFR émission d'une trame