Numéro d'ordre :...../

REPUBLIQUE DU SENEGAL

UNIVERSITE CHEIKH ANTA DIOP - DAKAR

ECOLE NATIONALE SUPERIEURE UNIVERSITAIRE DE TECHNOLOGIE

DEPARTEMENT INFORMATIQUE

Cycle : ingénieur

MEMOIRE DE FIN D'ETUDES

pour l'obtention du

DIPLOME D'INGENIEUR EN INFORMATIQUE

Sujet :

ETUDE ET REALISATION D'UNE INTERFACE INTEGREE SOUS WINDOWS 3 DE LA MESSAGERIE ELECTRONIQUE DU SYSTEME UNIX.

Présenté et soutenu publiquement le 5 juillet 1991

Par : Monsieur **Papa Abdou DIALLO**

sous la direction de Monsieur Hervé Chevillotte à l'ORSTOM

Année universitaire 1990/1991

S O M M A I R E

PARTIE II

LE PROGRAMME MESSOR SOUS UN DEVELOPPEMENT WINDOWS 3.0

PARTIE III

LE PROGAMME MESSOR

CES

REMERCIEMENTS

SONT DEDIES ...

REMERCIEMENTS

Au **P**ère tout **P**uissant
que Votre **LUMIERE ETERNELLE** et Votre **INFINIE GENEROSITE**
nous soient accordées

A mes Grands Parents : **in mémoriam**
A mon **Père** et ma **Mère** : jamais je ne saurais assez vous remercier et vous témoigner toute ma reconnaissance.

A mes frères et soeurs
A ma cousine Lissoune BA
A mon cousin Tafsir Momar Guèye
A mes oncles et tantes ; cousins et cousines
A Mlle Amoun Abdi Djiama merci d'avoir été à mes cotés pendant tout ce travail
Au Docteur Masserigne Soumaré, Merci pour tous les encouragements lors de la préparation du concours
A Mr Cheickh Tidiane Dia
A Mr Olivier Recy VSN à l'ORSTOM, merci d'avoir facilité mon intégration dans cet institut
A Mr Pape NDiaye du service de la reprographie de l'ORSTOM
A tous mes amis : ils sauront se reconnaître
A toutes mes amies du couloir Souleymane Seck
A Mme Elise Adamah,
A tous mes promotionnaires du cycle ingénieur
A tout le personnel du restaurant de l'ENSUT

A MES MAITRES ET PROFESSEURS

A tous ceux qui m'ont instruit dans les préceptes de l'informatique, particulièrement à:

Mr CLERCIN, nous reconnaissons en vous, une bonté sans réserve, une disponibilité engageante, un sens heureux de la pédagogie et de l'initiative ;

Mr Tidiane Seck, vous avez été pour nous plus qu'un enseignant, un grand frère, merci de votre disponibilité sans faille et votre sens de la mesure.

A **Monsieur Hervé Chevillotte**, nous apprécions à leur juste valeur Votre sens élevé des relations humaines, ce travail est le Vôtre, et nous sommes très heureux de reconnaître ce que nous vous devons.

A mes maîtres de l'école liberté I,
A mes professeurs du Lycée Blaise Diagne,
A mes professeurs de la facultés des sciences.

INTRODUCTION

I - INTRODUCTION

Les problèmes de téléphone, les notes égarées, les mémos délivrés trop tard, les messages perdus, l'oubli des confirmations.

Autant de maux qui peuvent nuire à une communauté pour laquelle la communication compte. En effet rares sont les individus qui travaillent seuls aujourd'hui. La communication est l'une des clés principales de réussites, bien communiquer , au bon moment, au bon destinataire..., est donc devenu une nécessité.

Les besoins en communication existent d'abord dans le cadre restreint d'un groupe de travail : ce sont les plus fréquents. Utiliser un réseau pour partager ou échanger de l'information à ce niveau apporte énormément de confort : plus besoin de courir d'un bureau à l'autre, certitude que le message sera reçu, même si le collègue n'est pas présent au moment où on cherche à le joindre. De manière moins systématique, les groupes de travail doivent communiquer les uns les autres.

Si les petits réseaux sont connectés entre eux pour former des inter-réseaux plus ou moins vastes, on étend le confort d'utilisation à toute l'entreprise. Il est souvent plus intéressant d'utiliser une messagerie que le téléphone, car deux appels sur trois au moins n'aboutissent pas.

Pourquoi une messagerie est-elle indispensable aux groupes de travail?

Les résultats d'une étude faite par Apple Magazine (France) en attestent.

Selon cette étude 85% des télécommunications passent par la voix, mais 75% des appels n'atteignent pas directement le correspondant et génèrent des messages téléphoniques.

91% des messages téléphoniques demandent un nouvel appel, 55% des appels ont des communications unilatérales (utiles à 44% , la conférence est nécessaire sur une messagerie). 60% des appels reçus sont moins importants que le travail qu'ils interrompent, 76% des appels ne sont pas urgents et pourraient être différés

Ces chiffres parlent d'eux-même.

En effet la probabilité de ne pas pouvoir joindre directement la personne avec laquelle on veut communiquer au moment où on le souhaite est loin d'être négligeable.

Pour une communauté scientifique comme l'ORSTOM (Institut Français de Recherche pour le Développement en Coopération), qui couvrent une dizaine de pays répartis sur trois continents, la messagerie électronique est un outils indispensable permettant d'améliorer la productivité personnelle et celle de l'Institut. Le Réseau Informatique de l'Orstom (R.I.O.) est doté en standard de tout ce qui est nécessaire pour un bon fonctionnement d'une messagerie électronique.

Le R.I.O. fonctionne sous le système d'exploitation UNIX. L'accès au serveur de messagerie existante s'effectue dans la plupart des centres ORSTOM avec un PC/AT sous PCNFS (*) et à travers un réseau Ethernet (*).

Certains logiciels existent déjà pour l'intégration de cette Messagerie sous PC/AT, mais n'offrent pas une bonne convivialité et on leur reproche surtout de manquer de certaines fonctionnalités.

Dans le cadre de notre stage en entreprise pour l'obtention du diplôme d'ingénieur en informatique, il nous a été proposé de développer une interface avec multifênetrage de la messagerie, afin d'offrir aux utilisateurs en plus des fonctionnalités de la messagerie électronique sous UNIX, le transfert de fichiers de petites tailles, une bonne ergonomie, et une assez grande ouverture pour de futurs développements.

Il s'agira aussi de mettre en place un système de messagerie efficace en tenant compte du modèle de référence ISO (International Standardization Organization) *(voir Annexe 1)*. En effet la fonction messagerie est une couche logiciel qui vient s'installer au dessus des autres couches de l'architecture réseau. Elle se situe au niveau de la couche application qui, peut contenir de très nombreux services qui peuvent être regroupés en deux catégories : les services utilisant le mode connecté et ceux utilisant le mode non connecté tel la messagerie. La messagerie s'appuie en fait sur les fonctionnalités offertes par un réseau de transport de l'information pour permettre la communication entre usagers.

De plus , dans un souci de faciliter l'échange international de messages, le CCITT a défini une norme (recommandations de la série X400) concernant les systèmes de messagerie. Cependant cela n'empêchera pas chacun de ces systèmes de posséder son propre format de messagerie, sa propre présentation et donc de pouvoir mettre en avant ses qualités intrinsèques face à la concurrence.

Nous allons dans la première partie de ce qui suit, donner une vue d'ensemble pragmatique de la messagerie dans son aspect purement technique mais aussi pour son intérêt économique et scientifique pour les pays en voie de développement et dans la deuxième partie présenter la solution retenue et les perspectives de son extension.

PREMIERE PARTIE

PRESENTATION ET ETUDE TECHNIQUE DETAILLEE DU RESEAU INFORMATIQUE ORSTOM

II.- LES SOLUTIONS MISES EN PLACE DANS LE R.I.O

II.1 ORGANISATION DU R.I.O.

Dans son principe le R.I.O. est constitué d'un noeud principal, de noeuds secondaires et de points d'accès, suivant une topologie en étoile. Chaque noeud est relié au noeud principal. Le transfert de l'information se fait point à point. Dans ce type de topologie le logiciel est centralisé ce qui présente un avantage au niveau de la gestion du R.I.O. qui est plus simple du point de vue économique, le coût étant réparti entre les différents sites. Mais cette topologie en étoile n'est pas très souple du fait qu'à chaque extension il faut une liaison supplémentaire ce qui à la longue risque de poser des problèmes de limite du noeud principal.

1.1 - Le noeud principal

Le noeud principal situé à Montpellier en France sert de relais entre les différents noeuds du R.I.O.. Tous les messages en provenance des autres noeuds et des points d'accès qui lui sont rattachés sont d'abord acheminés vers ce noeud principal avant d'être redistribués. Ce noeud se charge aussi de la liaison avec les autres réseaux (EARN et Fnet) : c'est en quelque sorte le commutateur qui établit des circuits entre paires de noeuds. C'est aussi le serveur principal. Il est équipé :

* de mini-ordinateurs (ou stations de travail), organisés en réseau local dont un est réservé exclusivement au R.I.O..

* de périphériques lourds (disques de grande capacité), dérouleurs de bandes (streamer 60 Mo), tables traçantes, tables à digitaliser).

* d'installations de télécommunication : lignes x25 (sur "TRANSPAC"), lignes RTC, lignes spécialisées vers d'autres sites (CNUSC)

* il est en service 24h/24h.

1.2 - Les noeuds secondaires

Ils concentrent les messages d'un pays et les transmettent par "lot" vers le noeud principal à Montpellier.

Les noeuds secondaires sont équipés :

* d'un ordinateur (station de travail ou micro-ordinateur)

* d'accès télécom au minimum : une ligne RTC. Une ligne directe réservée à cette usage offre plus de fiabilité. SENPAC offre cette possibilité en louant un modem qui sera totalement dédié au locataire ; alors que pour une ligne non directe les risques de saturation sont fréquents.

Le plus souvent les noeuds secondaires envoient, à des moments déterminés (au moins une fois par jour), les messages vers le noeud principal à Montpellier. Cette transaction peut se faire à leur initiative ou à celle du noeud principal.

1.3 - Les points d'accès

Les points d'accès n'ont aucun rôle actif, ils permettent de communiquer avec un noeud secondaire ou le noeud principal. Ce sont des "terminaux".

La liaison se fait à leur initiative. Plusieurs types de matériel sont utilisables :

* le plus performant : micro-ordinateur plus modem (MOdulateur-DEModulateur)

Le modem permet de franchir successivement les étapes suivantes :
- établissement du circuit
- initialisation (adaptation à la ligne)
- transmission
- libération.

Il permet à l'aide d'un logiciel de communication, d'envoyer des messages déjà préparés ou des rapports issus d'un traitement de texte ; ainsi que de recevoir sur le disque local des messages ou des fichiers.

* le moins cher : minitel 1B (gratuitement distribués en France aux abonnés au téléphone, elle sont loués par Senpac à leurs abonnés).

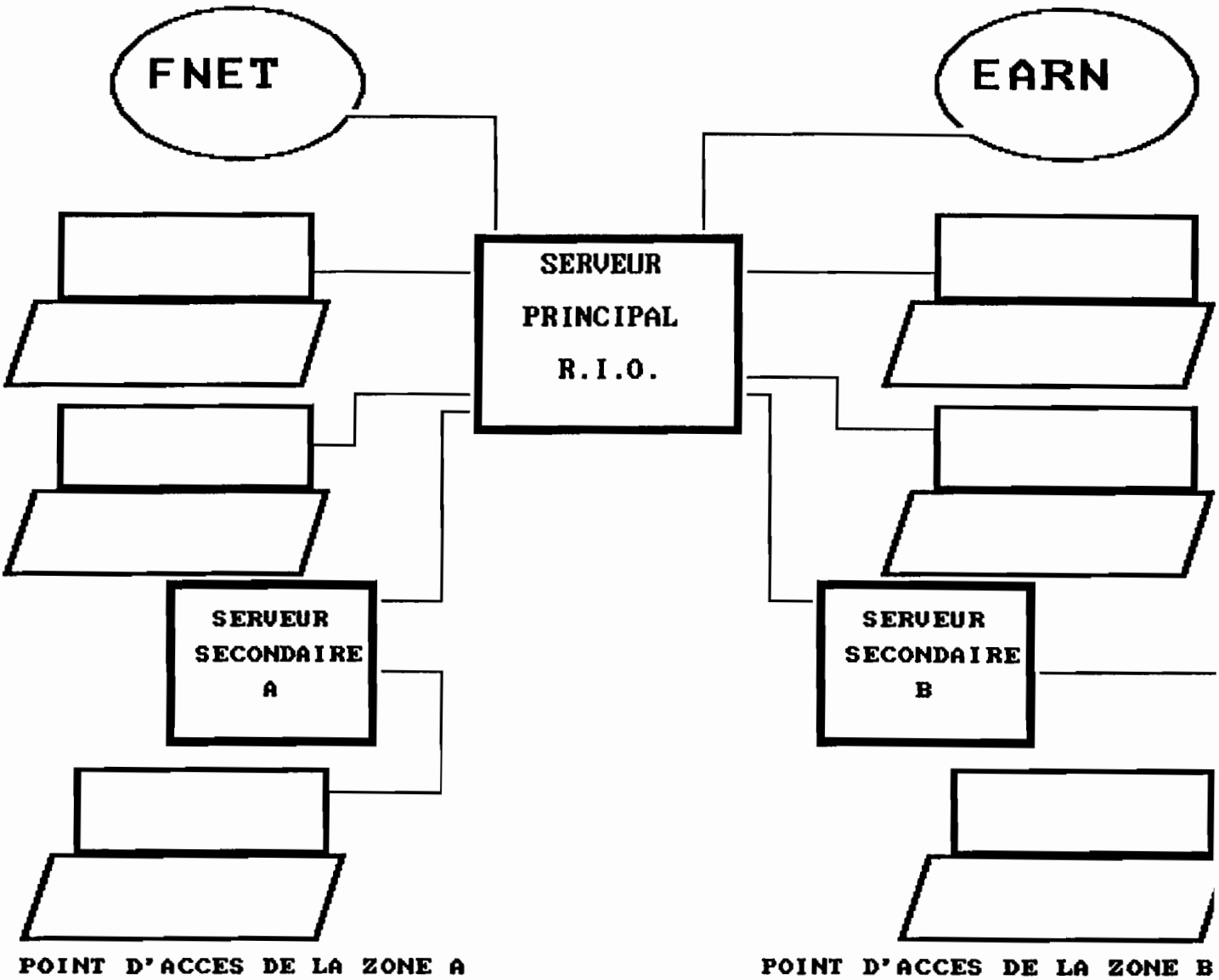


Figure : 1

SCHEMA GENERAL DU R.I.O. :

LE NOEUD PRINCIPAL

ET

ET LES NOEUDS SECONDAIRES

II.2 - DISPOSITIF TECHNIQUE

2.1 - LE SCHEMA GENERAL : UNE ORGANISATION EN ETOILE

Comme nous l'avons dit dans le paragraphe précédent le R.I.O (Réseau Informatique Orstom) est construit en étoile autour de l' ordinateur central installé à Montpellier. Dans chaque pays relié, un site distant communique directement avec ce site central.

Le transport des données se fait à travers les réseaux publics commutés tel que SENPAC (Réseau National de Transmission de Données à Commutation par Paquets) au Sénégal ou le Réseau Téléphonique Communté (RTC). Le R.I.O n'utilise pas de liaisons spécialisées. Il s'appuit sur les *protocoles* de transmission de données du système UNIX^{TM} : TCP/IP (fonctions terminal distant et transfert de fichier) et UUCP.(UNIX TO UNIX Communication Protocol).

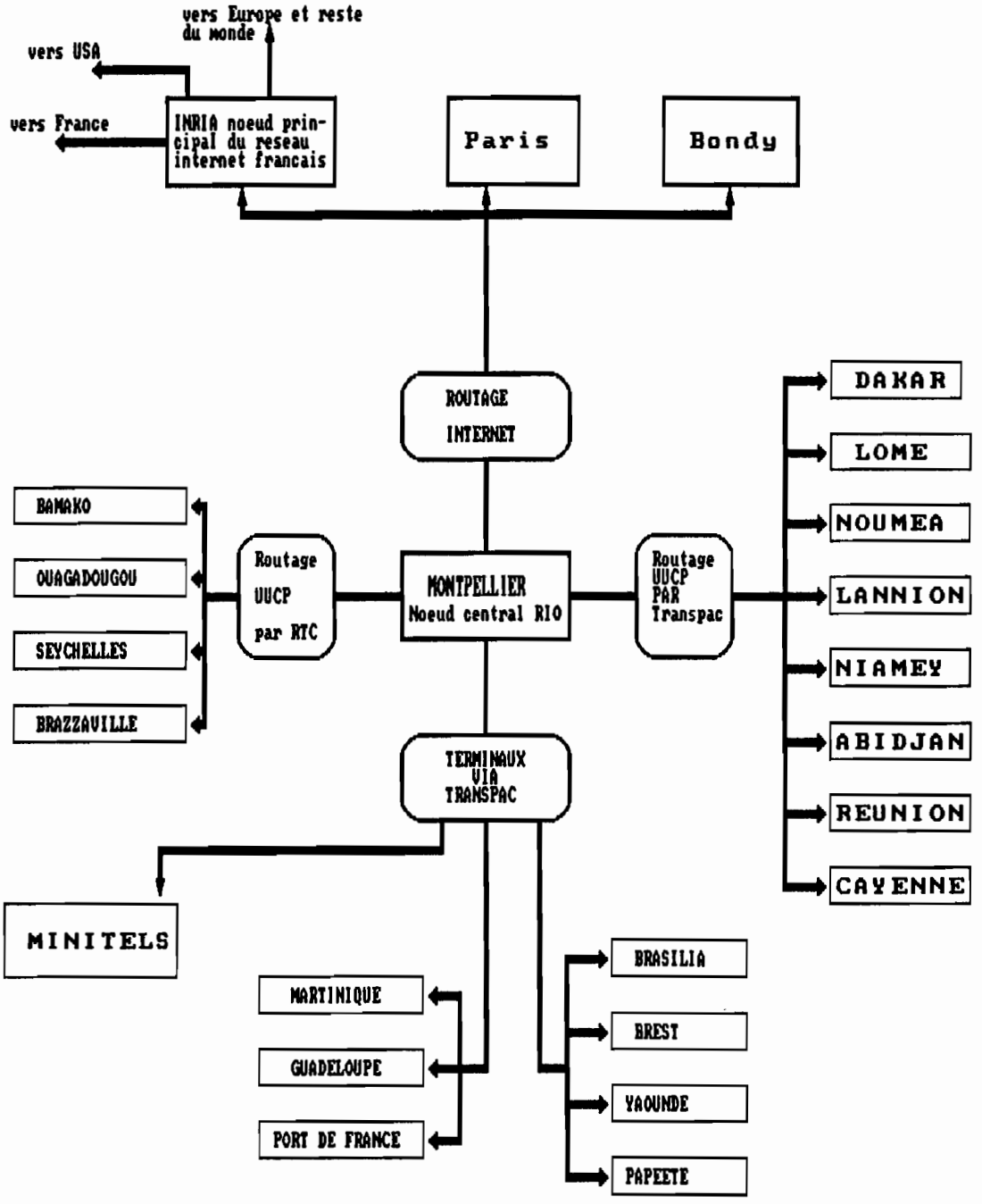


Figure : 2

SCHEMA GENERAL DU RIO : UNE STRUCTURE EN ETOILE.

2.2 - LE SCHEMA LOCAL

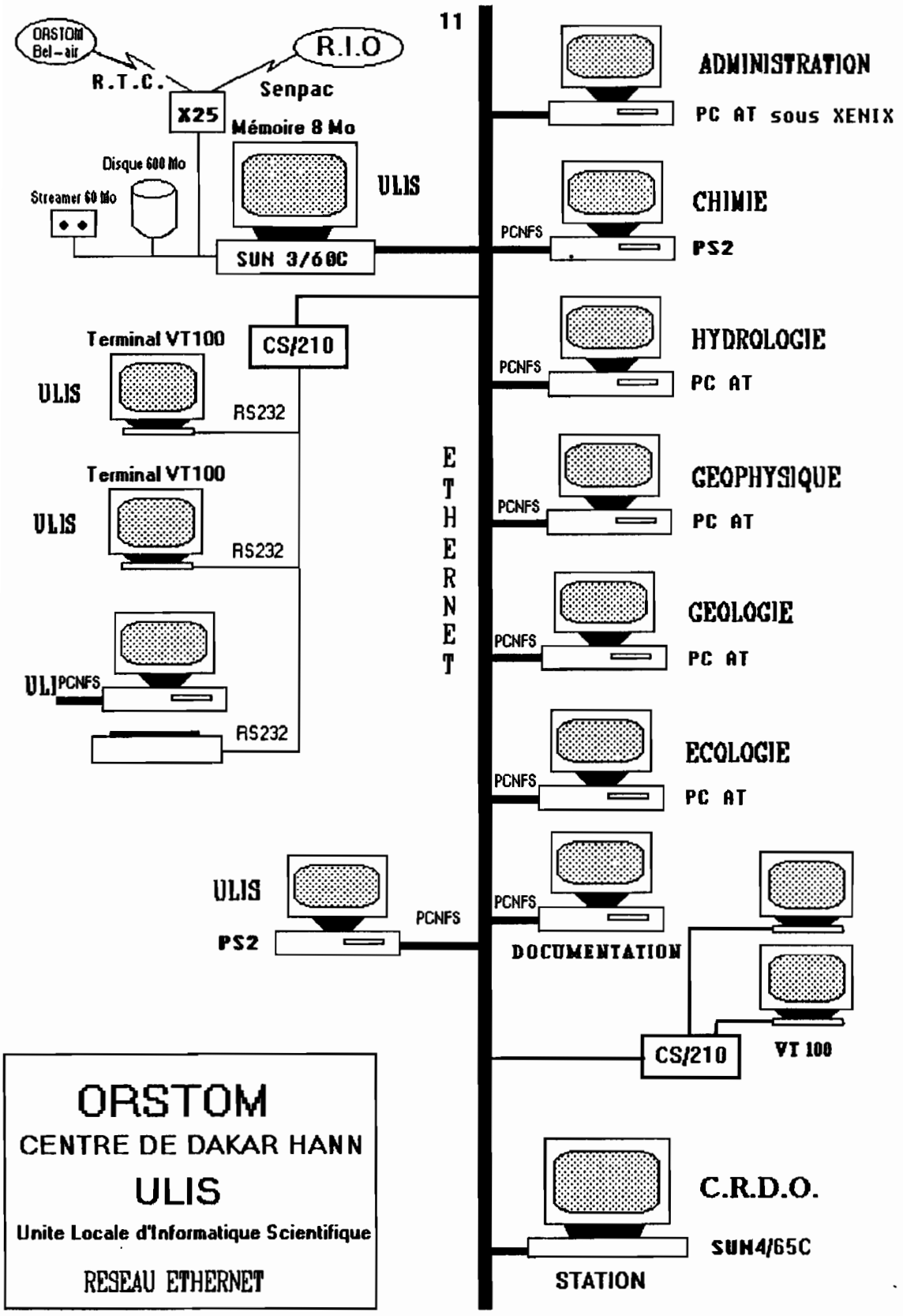
Constitués de mini-ordinateurs sous le système d'exploitation UNIX (principalement des stations de travail Sun-Microsystem TM), les réseaux locaux possèdent au moins une machine qui est à la fois serveur de communication et serveur de réseau local. Enfin, loin d'être dédié à ces deux fonctions "réseau", qui n'utilisent qu'une faible partie des ressources, ces ordinateurs assurent les services classiques de station de travail scientifique : calculs, traitements graphiques, bases de données...

Dans ce dispositif les PC et MAC, disposent des fonctionnalités d'un grand système tout en gardant leur caractère "convivial". L'accès au réseau n'est qu'une fonction supplémentaire pour les différents postes de travail individuel (compatible PC ou Macintosh).

Les services réseaux sont assurés par les *protocoles* Ethernet, TCP/IP et PCNFS^{TM} (serveur de fichiers et partage d'imprimante sous DOS). Les Macintosh sont intégrés à travers une passerelle (Gatorbox^{TM} de chez Cayman Systems^{TM}) réunissant le réseau principal (TCP/IP, NFS) et le réseau AppleTalk^{TM}.

Les logiciels qui servent d'interfaces entre les micros et le serveur UNIX, présentent ce dernier vis à vis des:

- Macintosh comme un serveur Apple-Share
- PC sous MS-DOS^{TM} comme un serveur NetBIOS^{TM}.


11
ORSTOM Bel-air
R.I.O
R.T.C.
Senpac
X25
Mémoire 8 Mo
Disque 600 Mo
Streamer 60 Mo
ULIS
SUN 3/60C
Terminal VT100
CS/210
ULIS
RS232
Terminal VT100
ULIS
RS232
ULI PCNFS
RS232
ETHERNET
ADMINISTRATION
PC AT sous XENIX
CHIMIE
PCNFS
PS2
HYDROLOGIE
PCNFS
PC AT
GEOPHYSIQUE
PCNFS
PC AT
GEOLOGIE
PCNFS
PC AT
ECOLOGIE
PCNFS
PC AT
ULIS
PS2
PCNFS
PCNFS
DOCUMENTATION
CS/210
VT 100
C.R.D.O.
SUN4/65C
STATION
ORSTOM
CENTRE DE DAKAR HANN
ULIS
Unite Locale d'Informatique Scientifique
RESEAU ETHERNET

III - CARACTERISTIQUES GENERALES DES SERVEURS

1 - Les ordinateurs

Comme nous l'avons déjà dit, ce sont des "machines UNIX", elles sont fournies par de nombreux constructeurs (parmi lesquels : IBM, SUN, Apple ...). Leur puissance de calcul peut être très variable et donc adaptée aux besoins de chaque utilisateur.

L'invariant c'est le système d'exploitation UNIX.

2 - Aspects du système UNIX

Il existe de nombreuses versions du système UNIX développées par A.T.T..
Le R.I.O. ne prendra en compte que les systèmes suivants :

- UNIX BSD 4.2 (dont SUN OS)
- UNIX BSD 4.3 (dont ULTRIX)
- UNIX Système V release 2, 3 et suivantes dont
  (SPIX, VENIX, XENIX).

Le système UNIX a plusieurs particularités dont, les plus importantes sont :

- multi-tâche,
- multi-utilisateur
- portabilité,
- bibliothèque d'applications,
- communication et son système de messagerie électronique
- ...

2.1 Système multi-utilisateurs et multitâches.

- **multi-utilisateurs :** en effet plus d'un terminal peuvent être connectés à la même machine simultanément pour exécuter des tâches différentes, le système se chargeant de la gestion des différentes requêtes émises par ces terminaux. Exemple plusieurs personnes peuvent envoyer ou recevoir simultanément des messages.

- **multi-tâches :** lorsque le système imprime des messages reçus on peut taper le texte des réponses aux messages ou trier une liste alphabétique d'adresse de correspondants. C'est comme si plusieurs machines étaient reliés entre elles pour exécuter séparément chacune de ces tâches.

2.2 La communication avec le système UNIX

Les protocoles de communication sont partie entière du système UNIX. Cette communication inclue les options suivantes :

- la communication entre différents terminaux connectés à la même machine.

- la communication entre utilisateurs d'une machine donnée et utilisateurs d'une autre type de machine.

- la communication entre machines de tailles différentes et reliées sur des distances allant de quelques mètres à plusieurs dizaines de kilomètres.

Les systèmes UNIX que nous avons cités peuvent donc ainsi communiquer pour échanger des données, gérer des protocoles de communication, faire circuler des messages et exécuter des programmes sous certaines limites.

2.3 La portabilité du système UNIX

Il est facile de modifier le code du système UNIX pour son installation sur une machine au lieu de réécrire un nouveau système pour celle-ci. Unix fonctionne bien sur plusieurs types de machines (gros ou petits). UNIX communique très bien avec les IBM PC sous MS-DOS (cas de ULIS de DAKAR) et les Macintosh.

Les AT sous UNIX sont capables de lire et d'écrire des disquettes DOS, échanger des fichiers sous TCP/IP. Et les Macintosh en réseau "Appletalk" peuvent utiliser un serveur SUN à travers une passerelle.

IV - LE R.I.O. ET LES VOIES DE COMMUNICATION

Les protocoles de communication utilisés par le R.I.O. sont le UUCP protocole F sur X25 et UUCP protocole G sur RTC pour les installations à l'étranger et l'IP (*) sur X25 en France.

Il s'agit d'utiliser, au mieux, les moyens de télécommunication disponibles dans les pays d'accueil en fonction des objectifs recherchés :

* fiabilité du système

* coût de fonctionnement minimum de manière à en autoriser l'accès à tous.

* délais de distribution garantis. Il faut, par exemple que le courrier électronique ait un délai de distribution qui souffre la comparaison avec la télécopie.

Lorsqu'il existe, le R.I.O. utilise le réseau X25. Mais il est quelque fois difficile ou coûteux d'établir une liaison directe vers un concentrateur, alors ils sont exploités par les accès RTC vers un PAD (*) public.

Avant de décrire les protocoles de communication mis en place par le R.I.O, nous allons présenter trois types de liaison utilisés par ce dernier.

IV.1 - liaisons asynchrones séries

La transmission asynchrone, encore appelée *start-stop*, réalise une synchronisation au niveau des caractères. C'est une des liaisons les plus communément utilisées pour relier un ordinateur et un terminal. La quasi-totalité des ordinateurs, micros ou gros systèmes, disposent d'une interface série.

1.1 - **Principe**

La transmission des données se fait sur deux lignes : l'une émet (Transmit Data), l'autre reçoit (Receive Data). Les caractères sont envoyés les uns après les autres sur la ligne à des instants indifférents.
Un élément "*start*" (un à deux bit(s)) est inséré au début de chaque caractère (voir figure IV 1). Il correspond à un "0" binaire, ou "espace". Un élément "*stop*" (un à deux bits) est également inséré à la fin de chaque caractère. Il correspond à un "1" binaire, ou "marque". L'élément *stop* est maintenu entre chaque caractère : il n'a donc pas de durée infini.

Ce protocole a été consacré par la norme RS 232 de L'EIA (Electric Industries Association) en 1969. Puis affiné par l'avis V24 du CCITT (*).

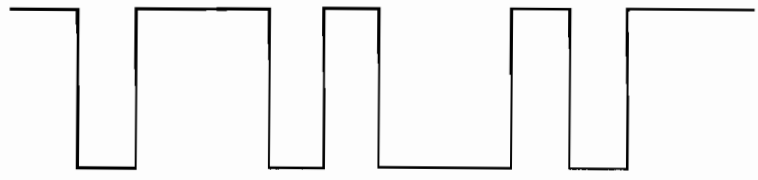


Figure IV 1

Ce type de liaison ne peut convenir que pour des courtes distances. La fiabilité dépend de la qualité de la ligne et de la distance. Pour effectuer des transferts de fichiers en toute sécurité, il est nécessaire d'utiliser un logiciel de transfert qui effectue des contrôles (*voir figure IV 2*).

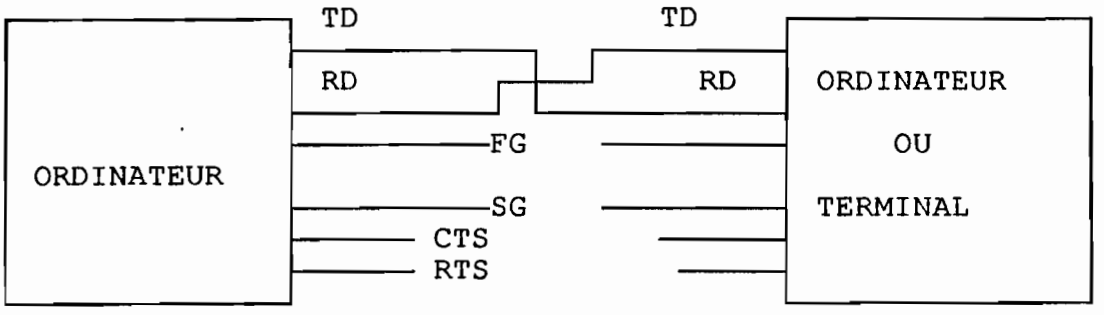


Figure IV 2

* signaux de données :
- Transmit Data (TD),
- Receive Data (RD),

* masses :
- Frame Ground (FG),
- Signal Ground,

et signaux de contrôle de flux (suivant l'application) :

- Clear To Send (CTS),
- Request to Send (RTS),
- Data Set Ready (DSR),
- Data Carrier Detect (DCD) (liaison avec un modem),
- Data Terminal Ready (DTR).

1.2 - Les protocoles de transfert de fichiers sur lignes asynchrones :

La technique <<start-stop>> est valable pour des terminaux lents, mais elle devient inefficace si on veut transmettre à vitesses plus élevées. Il repose en général sur le principe suivant :

Il n'y a ni <<start>>, ni <<stop>> entre deux caractères. Par contre, le flot de données est divisé en paquets, tous les bits étant transmis à intervalles de temps égaux.

Les deux systèmes, émetteur et récepteur, doivent donc être parfaitement synchronisés pendant toute la durée de la transmission d'un paquet. Cette synchronisation est commandée par le ou les caractère(s) de contrôle (check sum) fourni(s) par algorithme. Cette donnée de contrôle est recalculée lorsque le paquet arrive à destination. Et si le résultat n'est pas identique, un nouvel envoie de paquets est demandé à l'émetteur *(voir Annexe 1)*.

Des versions du logiciel KERMIT sur SUN, Appollo, PC, MAC, donnent de bon résultats sur ce type de liaison.

Pour répondre au besoin de communiquer entre utilisateurs, une série de protocoles permettent de relier deux à deux des machines utilisant UNIX sur ligne asynchrone et gérant la correction d'erreurs : UUCP (Unix to Unix communication Protocol), IP (Internet Protocol).

IV.2 - Liaison MODEM

MODEM : MOdulateur - DEModulateur. Cet appareil transforme le signal électrique carré (cf.asynchrone / V24) en un signal modulé dans une gamme de fréquences acceptables pour le réseau téléphonique commuté (celles de la voix humaine). Si le modem est *full duplex* (transmission simultanée dans les deux sens grâce à l'utilisation de deux bandes de fréquences), il est capable de gérer simultanément les données envoyées (Transmit Data) et les données reçues (Receive Data).

Le MODEM, comme nous venons de le présenter, est un dispositif capable de moduler le signal carré issu d'un autre dispositif qui transforme des signaux binaires transmis en parallèle, en signaux série de manière à ce que ce signal soit transportable sur lignes à grandes distances : l'entrée du modem se fait au protocole V24, la sortie en V21, V22, V22bis, V23 (*) ... suivant le cas.

Il existe pratiquement autant de types de modems qu'il y a de variétés de modulation en amplitude, en fréquence ou en phase on citera trois types de modem utilisé par le R.I.O.

- **MODEM à correction d'erreur** : De plus en plus, les fournisseurs de modem proposent des appareils intégrant la correction d'erreur. Il y a plusieurs niveaux de correction. Le premier est le signal : élimination des caractères parasites. Le second niveau est la correction des caractères altérés, dans ce cas le modem transmet des informations de contrôle comme les logiciels dont nous avons parlés plus haut.

- **MODEM à compression** : Pour compenser le surplus de transmission dû aux contrôles, les modems à correction sont en général pourvus de systèmes de compression. Ceux-ci permettent de gagner entre 20 et 50% de données. C'est évidemment très intéressant pour les liaisons longue distance.

- **MODEM de correction et compression** : Elles ne sont pas encore totalement stabilisées. Citons NMP5, ..., V24.

IV.3 - Le réseau SENPAC

SENPAC au Sénégal comme Transpac en France est un réseau national de transmission de données *par paquets.*

Crée il y a trois ans (1988), il est destiné à la communication entre ordinateurs, ou entre un ordinateur (serveur) et un terminal (utilisateur). Le Réseau SENPAC se conforme aux normes internationales émises par le CCITT ; il suit en particulier les avis X25 et X32 pour les accès synchrones, X3, X28 et X29 pour les accès asynchrones et X75 pour les liaisons internationales.

SENPAC et l'ORSTOM sont reliés par une liaison X25. La vitesse de transmission en mode synchrone est de 2400 bits/s.
Les caractéristiques de l'interface entre le serveur de l'ORSTOM et le modem sont celles définies dans l'avis X21 bis (25 broches, V.24). Les paquets sont ainsi préparés et transmis sous cette forme par le serveur.

La liaison X75 avec le noeud de transit international permet de toucher la quasi-totalité des réseaux publics de données à commutation de paquets. Ces réseaux couvrent actuellement déjà une grande partie du monde industrialisé, et opèrent actuellement une bonne percée en Afrique.

Des réseaux similaires sont en service dans la plupart des pays reliés au R.I.O.

3.1 - Principe des réseaux X25 :

C'est en 1976 que ce protocole a été consacré par le CCITT. Il contient en lui même les trois premières couches du protocole ISO. Nous développerons ici, trois aspects des réseaux publics X25.

3.1.1 Le matériel :

Des commutateurs ou concentrateurs sont reliés entre eux par des lignes spécialisées (deux ou quatre fils). Les PAD (Packet Assembly Disassembly) sont les "porte d'entrée" du réseau.

Les ordinateurs ou terminaux clients du réseau sont appelés des ETTD (équipement terminal de traitement de données). Il y a deux types d'ETTD, les serveurs munis d'un PAD, ce sont des ETTD-P, ils travaillent en mode paquet (protocole X25), et les terminaux (ou considérés comme tel), ce sont des ETTD-C, ils travaillent en mode caractère (protocole V21, V22, V23).

Sur SENPAC les ETTD-P (serveurs) sont des abonnés. Ils disposent d'un numéros dans le réseau qui leur permet d'être appelés. Les ETTD-C ne sont pas nécessairement abonnés et peuvent accéder à SENPAC par RTC (*) avec les numéros nationaux : (exemple :21 64 00 à 1200 bauds).

3.1.2 Les données :

Elles sont groupés par paquets (inférieur ou égal à 64 octets sur SENPAC). Chaque paquet est *étiqueté* avec une adresse de destination. De plus, des *paquets* de service permettent aux différents commutateurs ou PAD (*) de communiquer des informations sur l'état du réseau. Lors de leur passage dans un concentrateur ou un PAD, les paquets sont contrôlés. Si des erreurs incorrigibles sont constatées le paquet sera ré-expédié.

Les paquets provenant de plusieurs ETTD (*) sont *multiplexés* (transmis sur la même ligne). Inversement, si une ligne est encombrée ou défectueuse les paquets sont automatiquement déroutés sur d'autres voies. Il en résulte une grande fiabilité et une optimisation des circuits.

3.1.3 Le coût:

Dans un réseau privé il n'y a que des coûts fixes : installations et location des lignes si elles traversent des zones publiques.

SENPAC pratique une facturation assez indépendante de la distance, le monde étant découpé en quatre zones : Intérieur du Sénégal, groupe 1 (Afrique), groupe 2 (Europe), groupe 3 (reste du monde plus Grande Bretagne). En dehors d'un abonnement mensuel (44 000 FCFA pour une vitesse 2400 b/s), la facturation prend en compte le temps de connexion et le volume de données transmis ou échangé.

Exemple de facturation d'une communication internationale par SENPAC

Relation entre 2 abonnés raccordés à 2400 bits/s (LS *) pendant 10 minutes pour un volume de 70 koctets.
(exemple DAKAR - FRANCE) :

Taxe à la durée (6,5 FCFA) : 6.5 * 10 = 65 FCA.
Surtaxe (international groupe 2) (coeff 16) : 65 * 16 = 1040 FCFA .
Taxe volume (11 FCFA) : 70 * 11 = 770 FCFA.
Surtaxe (international) (coeff 8) : 770 * 8 = 6160 FCFA.
Taxe de la communication (sans les 7% de TVA) :
Taxe durée + Taxe volume = 7200 FCFA. HT

Cette tarification est variable d'un pays à l'autre.

V - LES DIFFERENTS SERVICES OFFERTS SUR LE R.I.O.

Les documents diffusés par la Mission Technique Informatique de l'ORSTOM (journal au sein de celui-ci), présentent en détail ces différents services offerts sur le R.I.O..

V.1 - LA MESSAGERIE ELECTRONIQUE

La messagerie est le service de base du R.I.O.. Elle permet de transmettre, en toute confidentialité, des textes de quelques lignes à plus de 50 pages. Les messages sont acheminés vers leur destinataire dans un délais variant, selon les destinations, de quelques secondes à quelques heures.

1.1 - Principe

Le texte est tapé sur un poste du réseau ou minitel. Il peut être rédigé à l'aide d'un traitement de texte ou saisi directement. Le destinataire reçoit le message dans une *boîte à lettres électronique* sur son poste de travail habituel (micro, mini-ordinateur ou minitel). Une fois lu, le courrier peut être conservé dans un fichier informatique et réexploité.

1.2 - Les correspondants

La messagerie permet de correspondre d'une part avec les agents de l'ORSTOM et de ses partenaires autorisés ayant ouverts une *boîte à lettres* sur un serveur du R.I.O. et d'autre part avec les abonnés des réseaux internationaux : EARN, Bitnet, Eunet, Internet (*) européen et américain, Uunet, Usenet... Au total le R.I.O. permet d'atteindre des centaines de milliers de correspondants dans les secteurs de la recherche, de l'enseignement supérieur et des industries de pointe.

1.3 - Les adresses électroniques

Chaque utilisateur dispose d'une boîte à lettres. Celle-ci est repérée par une *adresse électronique* unique sur l'ensemble des réseaux internationaux. De plus, un service (Département, équipe de recherche, administration, ...) peut disposer d'une adresse "fictive" correspondant à une ou plusieurs personnes désignées pour répondre au courrier.

1.4 - Identification de l'expéditeur

L'expéditeur d'un message est identifié par un mot de passe individuel qui est demandé lors de l'utilisation du réseau. Cette procédure garantit l'authenticité et la confidentialité du courrier. Les messages expédiés sont revêtus d'un "compostage" comprenant : le nom de l'expéditeur, son adresse électronique, la date, l'heure de l'envoi et un numéro d'identification.

1.5 - L'annuaire central du R.I.O.

Un annuaire général des boîtes à lettres de l'ORSTOM est géré à Montpellier sur la machine principale. Celui-ci couplé à un système de réexpédition automatique permet d'envoyer du courrier sans avoir à se préoccuper du lieu d'affectation de son correspondant.

En revanche, il n'existe pas d'annuaire de l'ensemble des utilisateurs des autres réseaux reliés au R.I.O..

1.6 - Les listes de distribution

Contrairement au courrier papier ou à la télécopie, un message peut être envoyé à plusieurs destinataires même si ceux-ci résident dans des lieux différents. Les utilisateurs ont la possibilité de constituer leurs propres listes de distribution pour simplifier les envois réguliers.

1.7 - La réexpédition

Il est possible, comme dans un service postal traditionnel, de faire suivre son courrier vers un autre lieu (une autre boîte à lettres). La ré-expédition automatique permet aussi aux chercheurs en congé ou en mission de continuer à communiquer avec leur équipe.

1.8 - Les accents

La messagerie de l'ORSTOM est basée sur des méthodes (protocoles RFC822) standards de messagerie, c'est une des raisons pour laquelle elle communique avec les réseaux internationaux. Ces protocoles d'origine américaine ne prennent pas en compte les caractères "accentués". En effet elles n'incluent à priori que les caractères purement ASCII (*) (7 bits).
Pour les utilisateurs de la langue française, l'absence d'accents est ressenti comme une contrainte pénible et nuit à l'intelligibilité du texte. C'est pourquoi notre solution cherchera comme pour les logiciels de messagerie déjà existants tels que PCMESSOR et NFSMESSOR (*) à implémenter des caractères accentués français.

La version du MESSOR que nous proposons assurera aussi une gestion automatique du classement chronologique des courriers envoyés et reçus.

V.2 - LA TRANSMISSION DE DOCUMENTS

Le service "messagerie " se limite à l'envoie de texte. Nous entendons, par transmission de documents l'envoi de rapports "balisés", c'est à dire préparés avec un logiciels de traitement de texte et contenant diverses polices et corps de caractères, des indications de styles, de justification, de dessins...

Le principe est le suivant : le document est codé dans un format intermédiaire et intégré dans un message, il sera reconverti à l'arrivée sous sa forme primitive. Cette technique permet de transmettre des rapports d'environ 50 pages, entièrement balisés enrichis de graphiques et de dessins réalisés avec un logiciel de traitement de texte standard (sur Macintosh ou pc compatible IBM, avec Windows...). Les documents peuvent être repris et modifiés par le destinataire disposant du même logiciel ou d'un produit compatible avec lui.

V.3 - LA TRANSMISSION DE FICHIERS

Il s'agit de transmettre des ensembles de données scientifiques ou de gestion destinés à alimenter un logiciel de traitement. Ces données se présentent rarement sous la forme de texte, c'est pourquoi la transmission de fichiers est un problème différent de la transmission de textes, de plus, elle exige une fiabilité totale, la moindre erreur risquant de perturber l'ensemble des traitements postérieurs.

Plusieurs méthodes de transmission de fichiers sont implémentées sur le R.I.O. : FTP(*) qui est réservé aux sites disposant d'une liaison à débit élevé ou moyen ; UUCP(*), en transfert direct ou par message, qui est disponible sur tous les serveurs, enfin KERMIT(*) qui permet en outre la transmission de fichiers à partir d'un micro-ordinateur relié par téléphone.

3.1 - LE FTP

FTP (File Transfert Protocole) est la méthode de transmission de fichiers des réseaux *internet* (*). Les sites métropolitains de Paris, Bondy et Montpellier constituent un réseau Internet. Celui-ci appartient au *domaine Internet* européen qui est lui-même relié au domaine américain.

3.2 - Le FTP et le réseau "NFS"

Les grandes universités des Etats Unis (Berkeley, MIT, Cornell...) ont mis en commun de grands moyens informatiques dans le cadre du réseau Internet de la "NFS" (National Science Fondation). Ce dernier est ouvert aux organismes de recherche. L'ORSTOM a obtenu l'autorisation (appelée "statut connect") d'utiliser les ressources de la "NFS". L'Institut peut à ce titre prélever des logiciels dans les bibliothèques des établissements de la "NFS" et utiliser ses lignes de télécommunication pour échanger des fichiers par FTP.

3.3 - Le UUCP

En dehors de la France métropolitaine, les serveurs R.I.O ne sont pas reliés entre eux par Internet. Le protocole utilisé est UUCP (Unix to Unix Communication Protocole). La liaison est établie en *temps différé* (une à quatre fois par jour). UUCP permet d'exploiter les réseaux publics de téléphone.

3.4 - Le UUencode

La transmission de fichiers par UUCP nécessite quelques connaissances du système Unix et donc, en général l'assistance d'un spécialiste. En revanche, une technique dérivée "UUencode" est utilisable par tous. Elle consiste à transformer le fichier en un message standard puis à l'arrivée à effectuer la transformation inverse.

Cette méthode est utilisable entre tous les noeuds du réseau pour la transmission de fichiers de petite taille.

3.5 - Le KERMIT

Kermit permet de transmettre des petits fichiers entre un micro-ordinateur relié par ligne téléphonique et un serveur. Son ergonomie le destine plutôt aux initiés.

V.4 - LES FORUMS

Le forum électronique est un outils de débat scientifique. C'est l'équivalent en terme de messagerie, de la "communication à plusieurs" sur le téléphone. Les contributions au forum se font par l'envoi de messages qui sont automatiquement distribués à tous les participants au forum.

Le forum est constitué par un groupe d'utilisateurs du R.I.O. ayant chacun une boîte à lettres. Leurs adresses sont enregistrées dans une liste de distribution. Le forum porte le nom de cette liste.

Chaque forum est placé sous la responsabilité d'un animateur. C'est lui qui en demande l'ouverture et qui est à même de répondre à toutes les questions concernant le sujet, les participants et la nature des contributions attendues.

Les contributions au forum sont enregistrées dans un fichier chronologique. La liste des forums du R.I.O. et le fichier chronologique de chaque forum peuvent être consultés par tous.

V.5 - LA CONSULTATION DES BASES DE DONNEES

5.1 - En mode conversationnel

Les sites ayant un accès SENPAC ou équivalent (TRANSPAC, NIGERPAC, TOMPAC, ...) peuvent consulter des bases de données en mode conversationnel. L'accès à une base de données nécessite, en général, l'autorisation du gestionnaire de la base. L'ORSTOM gère sur des ordinateurs intégrés au R.I.O., une base documentaire : "HORIZON" et des bases de données administratives

5.2 - En mode différé, par message à un "info-serveur"

Un "info-serveur" est un programme informatique qui répond automatiquement à un message qui lui est envoyé. Le message contient une requête celle-ci est interprétée et le résultat est retourné à l'envoyeur du message.

A titre d'exemple, les info-serveurs suivants sont en service :
- "annuaire@orstom.fr" transmet la dernière version de l'annuaire,
- "forum.serv@orstom.fr" donne la liste des forums.

Les info-serveurs sont utilisables à partir de tous les accès au R.I.O. (y compris extérieurs).

VI - LE SYSTEME DE TRAITEMENT DES MESSAGES DANS LE R.I.O

VI.1 - Principe de la distribution des messages

Les entités rencontrées dans le principe de la distribution des messages sont :

- l'Utilisateur (U) :
qui est soit un être humain, soit une application
qui agit soit en tant que émetteur, soit en tant que destinataire.

- l'Agent Utilisateur (UA):
qui aide à la préparation des messages , met en forme les messages reçus avant leur présentation à l'utilisateur, soumet les messages de l'utilisateur au MTA. En résumé ses fonctions sont destinées à fournir à l'utilisateur les outils nécessaires à la composition et à l'envoi de son message (cas de notre projet).

- l' Agent de transfert de messages (MTA):
L'Utilisateur lorsqu'il a fini de taper son message et suivant les directives de l'UA demande à ce dernier de l'expédier.

Le MTA est un ensemble de processus informatiques qui assure le relais des messages à travers les réseaux de transport (publics ou privés), et leur délivrance au(x) destinataire(s).

Le principe de fonctionnement est le suivant :

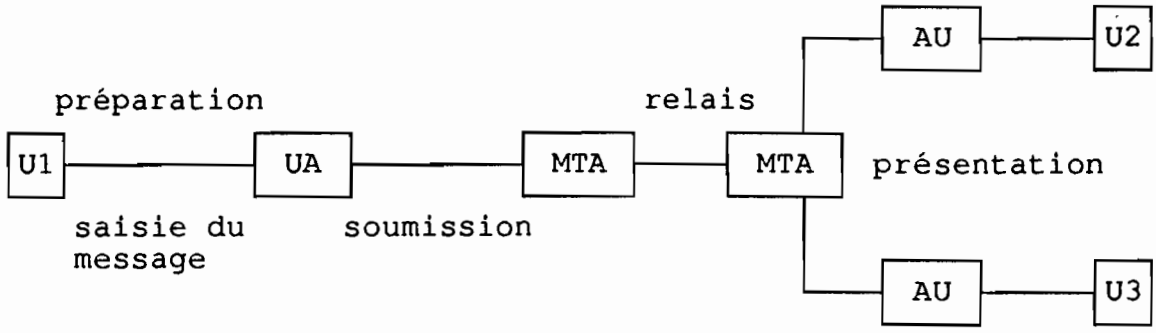


Figure VI 1

La structure d'un message étant la suivante :

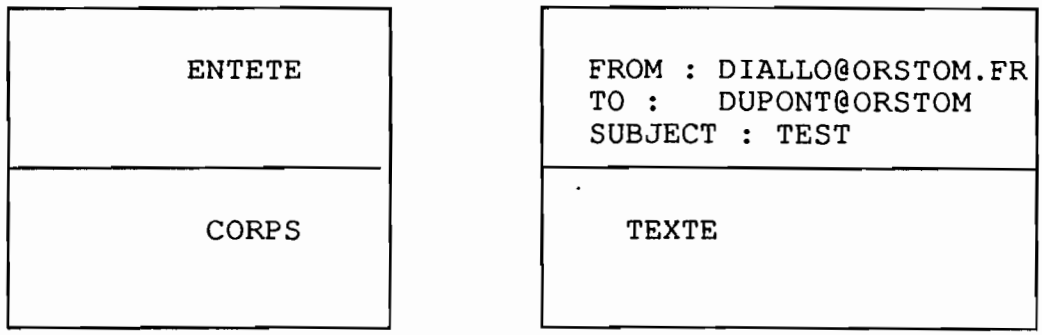


Figure VI 2

exemple de message

Dans le cas du R.I.O. un utilisateur accède par un équipement d'entre/sortie à l'UA pour créer, stocker des messages. Un message est émis par un utilisateur (u1), à destination d'un utilisateur (u2) à partir d'un point d'accès à un serveur (S1).

Deux cas peuvent se produire :

cas 1 : si u1 et u2 sont sur le même équipement (réseau local avec le même serveur S1) alors le message est enregistré et stocké par S1 et y restera jusqu'à ce que u2 le lise et le détruise (voir figure VI 3).

Ce serait la même démarche si le message était destiné à plusieurs utilisateurs du même site sauf que chacun d'eux a sa propre copie du message.

Echange de courrier entre deux utilisateurs sur le même équipement.

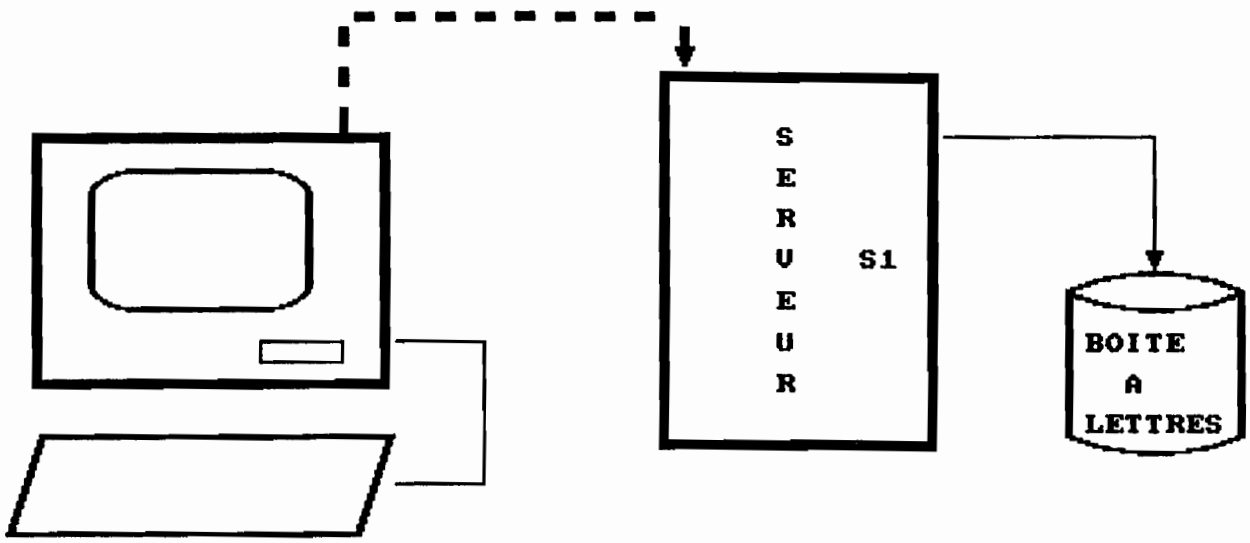


Figure VI 3

cas 2 : si u1 et u2 ne sont pas sur le même équipement, le message est d'abord stocké dans le serveur S1 de u1 et à une heure déterminée (ici on tiendra souvent compte du tarif heures creuses et éventuellement de l'encombrement) à Montpellier qui se chargera de faire le routage vers le serveur destinataire S2 qui le conservera jusqu'à ce que son propriétaire le lise et le détruise.

Pour le cas de plusieurs destinataires c'est le même principe que dans le cas 1.

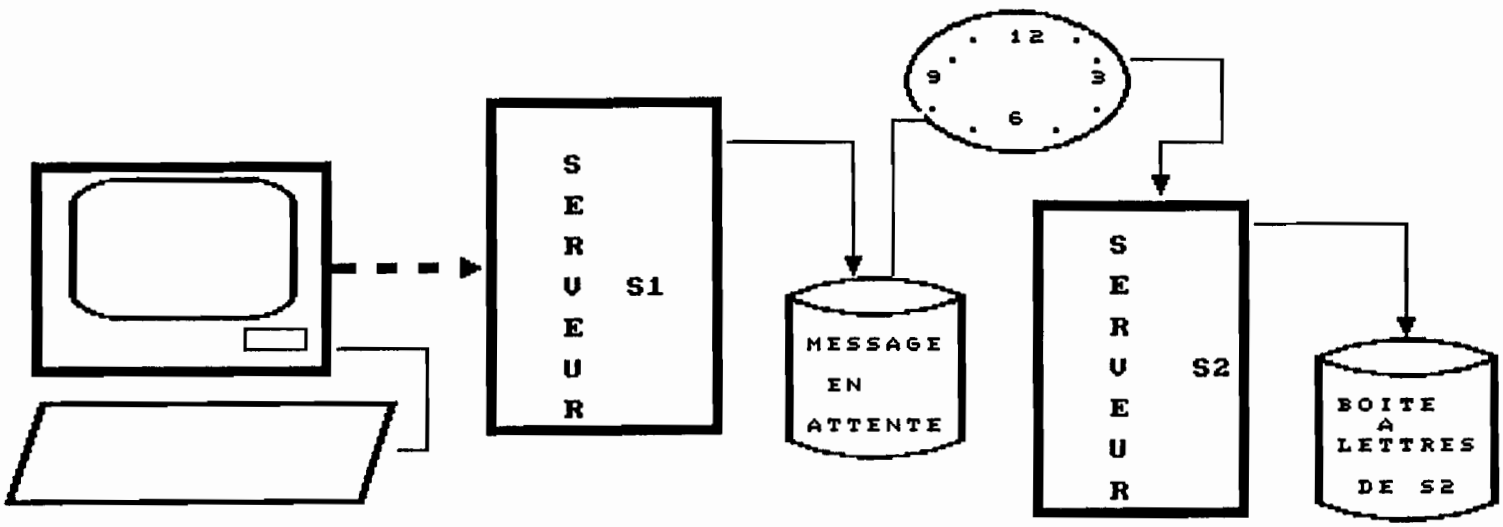


FIGURE : UI 4

VI.2 - Nommage et adressage

Au niveau de la gestion des adresses les *serveurs sont en général des serveurs de noms.*

Ainsi pour nommer un destinataire, on distingue deux noms :

- *nom primitif* donné par une autorité de nommage, il est unique et non ambigu mais non suffisant pour une identification unique et globale.

- *nom descriptif* qui est une liste d'attributs qui doit aboutir à une identification unique de l'utilisateur.

L'adresse permet une localisation géographique. C'est un nom descriptif.

A l'intérieur du R.I.O. la constitution des adresses se fait par le choix d'un certain nombre d'attributs. Ce sont :
- attributs personnels (nom patronymique...)
- attributs géographiques (pays, ville..)
- attributs structurels (nom de l'organisation ...)

Tous les agents de l'ORSTOM peuvent demander l'ouverture d'une boîte à lettres à leur nom sur leur site d'affectation et posséder ainsi une adresse.

La demande est faite directement auprès du responsable de site qui créera un "login" et donnera un mot de passe à l'agent si ce dernier est absent.

L'agent bénéficiera aussi d'une boîte à lettres sur le site de Montpellier, qui est le noeud principal du R.I.O..

Ainsi les correspondants se connaissent par leurs noms suivant le principe défini ci-dessus. Donc sans connaître l'adresse d'un destinataire on peut lui envoyer un message par la seule connaissance de son nom primitif. C'est le serveur principal qui, grâce à la gestion d'un annuaire qui constitue une aide pour les utilisateurs, transmet les messages vers le serveur destinataire.

C'est une des raisons pour lesquelles les messages transitent par le serveur principal.

Ce service est limité aux agents affectés à l'ORSTOM. Il n'est ni possible, ni souhaitable, qu'il couvre l'ensemble des correspondants de l'ORSTOM. Ils sont plusieurs centaines de milliers à pouvoir être joints par les réseaux internationaux. L'envoie d'un message à l'extérieur de l'Institut exigera donc d'indiquer l'adresse électronique complète (exemple pour le réseau Bitnet : diallo@orstom.uucp)

VII - L'INTERET DES RESEAUX POUR LES PAYS DU SUD

Nous ne pouvons terminer l'étude de cette première partie sans parler de l'intérêt que les réseaux informatiques pourraient avoir dans les pays du Sud.

Aujourd'hui, les réseaux informatiques se sont fortement développés au sein des pays industrialisés, dans des domaines aussi divers que l'économie, l'administration, l'éducation, les services publics et la recherche.

Sur ce dernier point, d'importants moyens sont déployés pour intégrer tous les réseaux des centres de recherche en un réseau national de la recherche et de la technologie (exemple de la France).

Les pays du Sud, surtout ceux de la francophonie, ne peuvent laisser passer cette opportunité et se doivent par conséquent, pour le développement de leur propre recherche, de développer en leur sein des structures de communication nationales et internationales capables de coopérer avec celles du Nord.

Dans les pays développés, le besoin n'est plus seulement de partager des ressources puissantes et volumineuses, mais d'offrir de nouveaux moyens de communication à l'échelle de la communauté scientifique internationale. De nombreux réseaux d'ordinateurs relient les chercheurs et les ingénieurs travaillant dans plusieurs milliers d'organismes. Par le biais de cette nouvelle forme de communication et de coopération, ces derniers peuvent échanger des données, communiquer des résultats ou demander la contribution d'un confrère sur tel ou tel sujet et ceci dans la plus grande sécurité au niveau de la transmission de l'information.

En revanche, les pays en voie de développement ne bénéficient que marginalement de cette explosion des réseaux. Cela tient évidement aux faibles moyens financiers de ces pays mais surtout à l'absence d'une industrie capable de produire une telle technologie.

Les taxes douanières effectuées sur ce matériel, ôtent tout espoir de voir une percée des réseau locaux dans les petites et moyennes entreprises et surtout dans les centres de recherches qui sont en fait les laissés pour compte dans ces pays.

Le sujet que nous allons développer ici, va montrer qu'au contraire, il est possible de développer un réseau informatique sur une grande échelle géographique en reliant des grappes de station de travail de coût modeste.

L'intérêt des réseaux informatiques pour les pays en voie de développement comme le Sénégal est indéniable. Le transfert technologique trouve ainsi la voie la plus courte et la plus rapide pour se concrétiser.

L'informatique grâce au bas prix des micro-ordinateurs, est maintenant diffusée dans tous les pays. Ce n'est pas pour autant que sa maîtrise technologique est généralisée.

D'après l'article de P. Renaud et M. Michaux, les laboratoires de "recherche-développement" sont les tous premiers utilisateurs des réseaux. Des banques de logiciels "domaine publique" ont été constituées par certains organismes pour permettre une large diffusion d'outils et de techniques nouvelles. La libre diffusion de ces logiciels et l'accès aux conseils de ceux qui les ont développés jouent un rôle important dans la diffusion du savoir informatique.

L'accès à ces outils, par la communauté scientifique et technique, est pour les pays en voie de développement un atout pour la maîtrise technologique de l'informatique. Le Sénégal, en créant un Ministère chargé de l'informatisation et de la modernisation de l'Etat, en a saisi tout l'enjeu pour son avenir.

On assiste par ailleurs à la mise en place dans nos pays de structures de communication comme SENPAC, et de centres de recherches comme l'ISRA (Institut Scientifique de Recherche en Agronomie) au Sénégal dans les domaines aussi variés que la santé, la nutrition, l'agronomie, la télédétection.

Grâce à ces structures, une collaboration avec les institutions internationales à travers les réseaux informatiques avec en toile de fond la messagerie (qui procure un dialogue permanent) pourrait être à l'origine d'une émergence d'un potentiel de recherche énorme dans ces pays.

Actuellement, les objectifs dans les pays en voie de développement sont plus proches de la satisfaction des besoins vitaux des populations (nutrition, santé éducation). La coopération avec des unités de recherche des pays développés travaillant sur ces sujets est très souhaitable. C'est le cas de l'ISRA et de l'ORSTOM au niveau du Sénégal. Grâce à SENPAC tous les centres de l'ISRA sont reliés à l'ORSTOM et entre eux.

Mais des difficultés liées à la constitution de réseaux informatiques retardent le développement de ces derniers dans nos pays. Elles sont de deux ordres :

- techniques, il y a le manque de personnel technique pour administrer des réseaux surtout ceux de la recherche et le manque de support technique (manque de fournisseurs capables de fournir l'appui nécessaire à leurs installations) et la défaillance de la maintenance du matériel proposé par les fournisseurs.

- financiers, inquiétude des responsables sur les risques de développement incontrôlable des dépenses de télécommunication (coûts souvent élevés des liaisons proposées).

Néanmoins vue la bonne coopération entre l'ORSTOM de DAKAR et l'ISRA, et les résultats que cela génèrent, tout porte à penser que le développement des réseaux locaux, reliés entre eux en inter-réseaux , serait une réponse au problème tant débattu du transfert de technologie.

DEUXIEME PARTIE

LE PROGRAMME MESSOR SOUS UN DEVELOPPEMENT WINDOWS 30.

VIII - GENERALITES

Rappelons tout d'abord les principales caractéristiques que l'application MESSOR doit posséder pour répondre aux objectifs qui nous étaient fixés :

- élaborer des messages et les envoyer,
- prendre en compte les messages reçus à chaque connexion au réseau (R.I.O.),
- consulter les boîtes à lettres,
- stocker les messages reçus dans un chrono (archivage),
- indiquer à chaque "login" l'état de la boîte à lettres,
- exploiter le contenu de la boîte à lettres (impression sauvegarde, destruction ...).
- gérer le répertoire de travail pour faciliter les transferts de fichiers textes par la messagerie.
- lister et modifier des fichiers textes.

Le premier problème qui s'est posé, c'est quelle solution pour une meilleure interface entre l'utilisateur et la machine ?

La création d'un menu déroulant et d'un système de multifenêtrage, capables de combiner à la fois des choix horizontaux, puis, à l'intérieur de ces derniers, de "dérouler" une petite fenêtre comportant elle aussi un ensemble de choix, exploitables d'un façon identique, répond aujourd'hui aux besoins des utilisateurs. Naturellement ces affichages doivent respecter le graphisme de l'écran en cours d'exploitation. Mais aussi cette opération, de création de menu et des fenêtres qui s'empilent, est devenue de plus en plus fréquente en informatique, particulièrement dans le domaine de l'"interfaçage" entre l'utilisateur et la machine.

Pour résoudre ce problème d'interface, nous avons imaginé un instant un utilisateur devant un écran comportant un menu. Ce dernier ne connaît pas parfaitement l'utilisation de ce logiciel ou simplement ne désire pas "ingurgiter" ou ne possède pas le "Bottin" qui est généralement fourni en tant

que manuel d'utilisation. Il va donc chercher à effectuer un choix en fonction de ce qui lui semble le plus probable.

Mais chacun peut avoir une idée différente de ce qui est plausible ou posséder des habitudes personnelles. Ainsi une personne trouvera tout naturel d'effectuer un choix en frappant au clavier la lettre majuscule en surbrillance qui figure au sein de l'option. Pour une autre il paraîtra évident d'utiliser les touches flèchées du clavier ou la souris afin de positionner le gros curseur en vidéo inverse sur l'option désirée, puis d'appuyer sur la touche <ENTREE> pour valider ce choix ou cliquer sur le bouton droit de la souris. D'autres utiliseront la barre espace, la touche tabulation etc ...
Ces quelques manipulations de touches couvrent certainement la quasi totalité des habitudes des utilisateurs, ce qui est largement suffisant.

Afin de satisfaire tout le monde, nous allons permettre l'utilisation de l'ensemble de ces options dans nos menus. Ainsi quiconque se trouvera placé devant nos écrans n'aura pas à se poser de questions inutiles quant à leur exploitation. Chacun y trouvera ainsi son compte car améliorer le dialogue entre l'utilisateur et la machine qu'il soit informaticien ou non, devient un critère de concurrence essentiel pour un logiciel.

Les offres de Windows qui sont assez intéressantes à ce niveau nous ont permis de faire une petite comparaison entre un développement en langage C, sous l'environnement DOS standard et l'environnement Window. Notre choix se portera finalement sur Windows non pas seulement pour les avantages qu'il offre par rapport à l'environnement standard DOS, mais aussi par la disponibilité de ce produit sur les postes de travail de l'ORSTOM de Dakar.

Nous feront ressortir les différents avantages lors de la présentation de la programmation sous Windows.

IX - POURQUOI LE CHOIX DE WINDOWS ?

Window 3.0 a été proposé en Mai 1990 sur le marché du logiciel. Il apporte beaucoup de changements par rapport aux versions antérieurs 1.01, 2.0, 2.1.

Windows a la réputation d'être facile pour les utilisateurs mais très difficile pour les programmeurs. En effet un novice en Windows y rencontre "d'étranges" concepts et il faut une réorientation complète par rapport à la manière habituelle de programmer. Cette phrase du célèbre programmeur en Windows Charles Petzold de MICROSOFT illustre bien les difficultés qu'on peut rencontrer quand on programme pour la première fois sous Windows.

"If at first you find Windows programming to be difficult, awkward, bizzarely, convoluted, and filled with alien concepts, rest assure that is a normal reaction. You are not alone." (PETZOLT).

Tout au long de notre exposé nous essayerons de soulever ces "difficultés" et "bizarreries" que nous avons rencontrées.

Les programmes Windows sont généralement écrits en C.
Pour construire la plupart des applications qui s'exécutent sous WINDOWS, il nous faut les outils suivants :
- Microsoft C Optimizing Compiler : CC ou le Quick C : QCL.
- Microsoft Segmented Executable Linker : LINK.
- Microsoft Windows Ressource Compiler : RC.
- Microsoft Windows SDKPaint : SDKPAIN.
- Microsoft Windows Dialog Editor : DIALOG.

Il faut aussi noter que les applications windows peuvent être développées sous les langages Pascal ou Assembleur. Mais ces langages obligent le programmeur à réécrire ou ajouter des fonctions Windows qu'un programmeur en C n'a pas besoin.

Généralement écrites en C, les applications sous Windows, sont pour plusieurs raisons très différentes de la façon de programmer en C standard :

- pour tourner efficacement sous l'environnement Windows, une application doit "coopérer" avec ce dernier et éventuellement avec d'autres applications.

- l'application doit retourner le contrôle à Windows chaque fois que c'est possible et partager toutes les ressources du système avec Windows et les autres applications.

La version 3 de Windows offre plusieurs fonctionnalités que l'environnement standard DOS n'a pas. Ainsi les applications Windows sont dans un certain sens beaucoup plus complexes que les programmes sous DOS.

Pourquoi ce choix de WINDOWS ?

IX.1 Comparaison avec l'environnement DOS

Comme nous l'avons dit ci-dessus, Windows offre beaucoup plus de possibilités que l'environnement DOS.
Ceci est compréhensible si l'on considère qu'en plus des routines pour construire des menus déroulants, des boîtes de dialogues, des barres de contrôles (ascenseurs) et d'autres types de fenêtres, Windows inclut :

- un GUI (Graphical User Interface featuring Windows) qui permet de faire les menus, les boîtes de dialogues, les contrôles etc..
- une file d'attente pour les entrées
- un DIG (Device Independant Graphics).
- le multitâche (Multitasking)
- DIA (Data Interchange between Apllications).

Pour développer une application en langage C sous l'environnement DOS, la plupart des programmeurs utilisent la bibliothèque standard de C "run-time library" pour procéder aux opérations d'entrées/sorties, à la gestion de la mémoire et tant d'autres activités.

La bibliothèque C assure les opérations standards de l'environnement DOS telles que les entrées par claviers (la fonction **getch()**), et un accès exclusif à la mémoire de l'ordinateur comme pour les entrées/sorties.

Sous Windows, ces affectations ne sont plus valables. En effet les applications Windows se partagent les différentes ressources de l'ordinateur y compris le CPU (Unité Centrale de Traitement) et communiquent avec les utilisateurs à travers une visualisation graphique, un clavier et une souris.

La section suivante va décrire, point par point, quelques unes des grandes différences entre Windows et DOS.

1.1 L'interface avec l'utilisateur

L'un des principaux buts visés par Windows, est de procurer un accès visuel à la plupart sinon à toutes les applications à la fois. Dans l'environnement multi-tâche, il est important de pouvoir donner à chaque application une partie de l'écran ; ce qui permet à l'utilisateur d'être en "conversation" avec toutes les applications. Certains systèmes font cela en donnant la possibilité à un programme d'utiliser tout l'écran, alors que d'autres attendent en "Background".

Sous Windows toute application a accès à une partie de l'écran et à n'importe quel moment.

Une application partage l'écran en utilisant une "fenêtre" pour "converser" avec l'utilisateur. Techniquement, une "fenêtre" c'est un peu plus qu'une partie rectangulaire de l'écran qu'utilisera une application. En réalité c'est une combinaison d'un dispositif d'affichage, tels que :

- les menus,
- les contrôles,
- les "scrolls bars" (ascenseurs),

que l'on utilise pour communiquer avec l'application.

Sous l'environnement DOS, le système prépare l'écran pour l'affichage automatique de notre application dès son démarrage. De manière typique, il fait cela en communiquant un programme de gestion de fichier à notre application.
Ainsi nous pouvons utiliser ce programme pour envoyer des sorties sur l'écran grâce aux fonctions standards du C ou les appels systèmes du DOS.

Sous Windows, on doit créer notre propre système d'affichage (fenêtre) avant de pouvoir faire une quelconque opération d'entrées/sorties. Une fois notre fenêtre créée, Windows produit une grande panoplie d'informations concernant l'utilisation future de celle-ci. Windows exécute automatiquement la plupart des tâches demandées par l'utilisateur telles que déplacement et dimensionnement de la fenêtre.

Un autre avantage de développer sous Windows une interface par rapport au C standard sous DOS est, que contrairement à ce dernier, qui accède à une seule fenêtre à un moment donné, l'application Windows peut créer et utiliser un nombre indéterminé de fenêtres (chevauchement de fenêtres), pour afficher ou recueillir des informations. Windows gère pour nous l'écran, contrôle le positionnement et l'affichage des fenêtres, et s'assure que deux applications n'ont pas accès au même instant à la même portion d'écran.

1.2 La file d'attente des Entrées

L'une des plus grandes différences entre une application Windows et un programme sous l'environnement C standard est la voie par laquelle ils reçoivent leurs entrées.

Sous DOS, le programme lit à partir du clavier en faisant explicitement appel à une fonction telle que **getchar**. Cette fonction attend que l'utilisateur appuie sur une touche avant de retourner le code du caractère tapé au programme. Windows, quant à lui, reçoit toutes ses entrées à partir du clavier, de la souris et de l'horloge , et les place dans la file d'attente de messages de l'application concernée.

Quand l'application est prêt à extraire les données, elle lit simplement l'entrée suivante dans la file d'attente de messages.

Sous l'environnement DOS aussi, un caractère saisi au clavier est codé sur 8 bits. Les fonctions standards du C ***getchar***, ***fscanf*** lisent un caractère à partir du clavier et retournent un code ASCII ou un autre code correspondant à la touche appuyée. Un programme peut aussi intercepter des interruptions d'autres dispositifs tels que la souris, l'horloge pour utiliser les informations recueillies comme des entrées.

Une application sous Windows reçoit ses entrées sous la forme de "message d'entrée". Ce message d'entrée Windows contient des informations qui dépassent le type d'information disponible sous DOS.
Il spécifie entre autre, la position de la souris, l'état du clavier, le code de la touche (si une touche est pressée), le bouton de la souris qui a été cliquée. Par exemple, il y a deux types de messages pour les entrées au clavier. Le WM_KEYDOWN et le WM_KEYUP qui correspondent respectivement à l'état d'une touche enfoncée et d'une touche relâchée. Windows produit pour chaque message venant du clavier un "device-independant virtual-key code" qui identifie la touche. Les entrées venant du clavier, de la souris ont tous le même format et sont traitées de la même manière.

1.3 Le DIG ou Device-Independant graphics

Sous Windows on a accès à un riche ensemble de dispositifs indépendants de graphismes. Ce qui signifie qu'une application peut facilement dessiner des lignes, des rectangles, des cercles ainsi que des figures très complexes. Parceque Windows offre un dispositif indépendant pour le graphisme, on peut utiliser les mêmes fonctions pour tracer un cercle sur une imprimante matricielle à point ou une à très haute résolution graphique.

Windows requiert un module de gestion de périphériques pour convertir les graphiques permettant de les visualiser correctement à impression, sur les traceurs, à l'écran ou toute autre type sortie. Ce module est un fichier spécial exécutable qu'une application peut charger et raccorder à un port de sortie ou tout autre dispositif de sortie.

Un "device context" représente le module de gestion de périphériques, le dispositif d'entrées/sorties ou parfois le port de communication. Notre application, exécute les graphiques à l'intérieur du "contexte" d'un dispositif spécifique.

1.4 Système multitâche

Windows est un système multi-tâches : plus d'une application peuvent s'exécuter en même temps. Tel n'est pas le cas sous l'environnement DOS standard. Les programmes écrits pour l'environnement suppose d'une manière typique qu'ils ont le contrôle exclusif des différentes ressources de l'ordinateur telles que les Entrées/Sorties, la mémoire, l'écran et aussi l'unité centrale de traitement (CPU). Cependant, sous Windows, les applications se partagent ces précieuses ressources lorsqu'elles s'exécutent en même temps. Pour ces raisons, Windows prend prudemment le contrôle de ces ressources. Et demande aux applications windows d'utiliser un programme d'interface spécifique qui garantie à windows son contrôle sur ces ressources.

Par exemple, sous DOS un programme, lorsqu'il s'exécute, a accès à toute la mémoire non utilisée : par le système, le programme lui même, ou les programmes résident (TSR terminate-but-stay-resident). Ce qui signifie qu'un programme à la liberté d'utiliser toute la mémoire disponible et comme il le veut.

Sous Window, la mémoire est une ressource partagée. Puisque plus d'une application peuvent s'exécuter simultanément, chacune doit coopérer avec les autres pour le partage de la mémoire afin d'éviter sa saturation. A une application, n'est allouée que ce dont elle a besoin, pour s'exécuter.

Windows ne fournit que deux types d'allocations mémoire :

- "global mémoire" pour de grosses allocations
- "local memory" pour les petites allocations.

Pour avoir une utilisation efficiente de la mémoire, Windows déplace ou libère souvent des blocs de mémoire. On ne peut pas être sûr que les objets auxquels on alloue une place mémoire restent à cette place surtout si plusieurs applications sont en cours d'exécution.

Un autre exemple d'une ressource partagée est le système d'affichage. Sous DOS le système octroie à notre application l'utilisation exclusive de son système d'affichage, avec la possibilité de changer la couleur du texte, du fond de l'écran et de passer du mode vidéo texte au mode graphique. Sous Windows, notre application partage le système d'affichage avec les autres applications, ainsi nous n'avons aucun contrôle sur l'écran.

IX.2 - Le modèle de programmation Windows

Ce modèle, basé sur les caractéristiques de Windows développées ci-dessus, nous a permis de réaliser l'application **MESSOR** (pour Messagerie ORSTOM).

Nous décrirons dans un premier temps certains outils qu'offre Windows et que nous avons utilisés pour la mise en oeuvre de cette application ; dans un deuxième temps les fonctions de **MESSOR** et enfin nous présenteront le programme MESSOR dans ses grandes lignes.

2.1 Ossature d'une application Windows

La plupart des applications Windows utilisent les éléments suivants pour "converser" avec les utilisateurs.

- Windows (fenêtres)
- Menus (menus)
- Dialog Boxes (fenêtres de dialogue)
- Message loop (Boucle de gestion de "messages" (entres/sorties))

2.1.1 Les fenêtres ou écrans

Une fenêtre n'est en faite qu'un dispositif d'entrées/sorties pour toute application Windows. C'est le seul chemin d'accès à l'écran d'une machine pour une application Windows donnée. c'est la combinaison de :

- barre du titre de la fenêtre
- barre des menus
- barre de défilement
- bordures
- d'autres caractéristiques pouvant occuper une partie de l'écran. *Voir Fig : 1, Annexe 2.*

C'est à sa création que l'on spécifie les caractéristiques du type de fenêtre qu'on veut avoir à l'écran. Windows, dessine ainsi pour nous la fenêtre et la gère en collaboration étroite avec notre application.

Pour gérer cette collaboration, Windows avise chaque fenêtre avant de procéder aux changements qui l'affectent. A cause de cela, à chaque fenêtre correspond une "fonction window" qui reçoit tous les "messages" appartenant à celle-ci, et exécute toutes les actions qui doivent être menées sur elle.
Voir MESSOR.C.

2.1.2 Les Menus

Les menus sont les principaux moyens pour l'utilisateur de faire exécuter des tâches bien déterminées à une application.

C'est un nom et une liste de commandes que l'utilisateur peut visualiser pour y faire des choix.

Une application possède un nom pour le menu et des noms de commandes à exécuter. Leur implémentation et leur affichage sont gérés par Windows.

C'est windows qui aussi, envoie des "messages" aux fonctions window de l'application quand l'utilisateur fait un choix. Le "message" étant simplement un signal émis par l'application pour l'exécution d'une tâche spécifique.
Voir MESSOR.C et Fig : 2, Annexe 2.

2.1.3 Les fenêtres de dialogue ou "Dialog Boxes".

Une fenêtre de dialogue est une fenêtre qu'on peut afficher temporairement pour permettre à l'utilisateur d'avoir plus de renseignements sur une commande, ou d'en fournir pour l'exécution des tâches suivantes

Un "Dialog Box" peut contenir un ou plusieurs "contrôle(s)". Un contrôle étant une toute petite fenêtre qui permet de saisir ou afficher une ligne de commande.

Par exemple "edit control" est une simple fenêtre qui permet à l'utilisateur d'entrer ou d'afficher un nom de fichier.

Un controle dans un "Dialog Box" permet de choisir une option, de saisir un mot de passe ....
Voir MESSOR.DLG et Fig : 3,4,5 Annexe 2

2.1.4 La boucle de message

L'une des principales caractéristiques d'une application Windows c'est aussi la "boucle de message" ou Message Loop qui gère les "messages".

Dés qu'une application reçoit des "messages" en entrée par l'intermédiaire du gestionnaire de la file d'attente des applications, cette boucle extrait les "messages" et les envoie aux différentes fenêtres concernées.
Voir Fig : 6,7 Annexe 2.

IX.3 - La bibliothèque Windows

Comme le C standard, Windows a sa propre bibliothèque. C'est à plus de 550 fonctions ou "messages" qu'une application peut faire appel. Ce qui veut dire qu'on ne pourra jamais mémoriser la syntaxe de ces fonctions ni tenter à travers cette application de donner la syntaxe de celles utilisées. Les lecteurs pourront se référer à l'importante documentation de Microsoft (voir bibliographie). On précisera seulement que toutes ces fonctions se trouvent dans le fichier d'inclusion **windows.h**.

Contrairement à la bibliothèque C, celle de Windows possède un DLLs (Dynamic-link-librairies) que le système attache à notre application lors du chargement de celle-ci. C'est une importante caractéristique de Windows car le DLLs diminue la taille du code exécutable.

X - Construction d'une application Windows : MESSOR

X.1 - Principe

La création d'une application Windows nécessite six phases. (*Annexe 2 Fig 7*)

1) la création du fichier source C qui contient la fonction **WinMain** (point d'entrée d'une application), les fonctions Windows, et tout autre code intervenant dans l'application.(Toutes les fonctions seront détaillées par la suite). *Voir MESSOR.C.*

2) l'utilisation des éditeurs de ressources tels que (SDKPAINT, Dialog Editor, Font Editor..) pour créer les icônes, les curseurs, les fenêtres de dialogues,... dont a besoin l'application. Elle fournit le fichier .DLG (comme dialogue) ou ICO (comme icône). *Voir MESSOR.DLG.*

3) la création du fichier d'information (script resource (.RC)) qui définit toutes les ressources de l'application. Il nomme les ressources qui sont créées dans le fichier (.dlg). IL définit aussi les menus et d'autres types de ressources tels que les accélérateurs (combinaison de touches). Voir *MESSOR.RC.*

4) la création du fichier contenant les modules de définition (.DEF). Ce fichier définit les attributs de chaque module de l'application tels que l'attribut des segments, taille des piles, les fonctions exportées. *Voir MESSOR.DEF*

5) utilisation d'un compilateur Microsoft ici le Quick C. La commande QCL est utilisée avec plusieurs options. Les caractéristiques de notre machine de travail (un IBM PS2 modèle 50 Z) nous ont conduit, lors de l'installation des outils de développement Windows, à choisir l'option suivante de compilation. : qcl -c -AM -Gws -Oas -Zpe .

-c pour supprimer l'éditeur de lien du C standard,
-AM pour spécifier que c'est le "Medium memory model"
-Gws pour l'ajout des fonctions exportées de Windows
-Oas pour optimiser la taille du code exécutable
-Zpe pour produire des informations sur le nombre de lignes
et condenser les structures.

Le "Link" ou édition de lien se fera avec l'option /NOD qui signifie **/nodefaultlibrarysearch** (empêche l'éditeur de lien d'utiliser la bibliothèque C). /NOD étant une abréviation.

6) l'utilisation du RC (Resource Compiler) pour compiler le fichier (.RC) et l'ajouter au fichier exécutable (.exe)

X.2 - Généralités sur les menus de MESSOR

La philosophie Windows (fig : 6) nous a suggéré une démarche pour la mise en oeuvre de l'application Messor. Dans la première partie de ce mémoire, lors de l'étude de la messagerie sur le R.I.O., nous avons recensé des entités telles que : message, annuaire, chrono, destinataire, et expéditeur. Par la suite expéditeur et destinataire sont tout simplement devenus utilisateur, car ce dernier peut en fait recevoir ou envoyer des "messages".

Aujourd'hui, tout logiciel de messagerie doit permettre au minimum de "saisir" des "messages", de les stocker, de les imprimer et de les diffuser.

Nous avons dans les menus ***Message***, ***Services*** et ***Edition*** des options de commande qui permettent de gérer les quatre fonctionnalités citées ci-dessus.

D'abord pour la saisie de "messages", nous avons crée une fenêtre éditeur ("*edit*" sous windows). Le menu ***Edition*** offre plusieurs options *couper, coller, copier, annuler, sélectionner*) pour la mise en forme de son texte comme dans un traitement de texte classique. Les options "*nouveau message*" du menu Message et *Répondre* de la fenêtre de dialogue qui permet d'afficher les "messages" reçus permettent d'effectuer cette saisie.

Pour l'impression, l'option *Imprimer* du menu ***Services*** se charge des "messages" reçus alors que l'option du menu ***Message*** permet l'impression des "messages" tapés et celle des fichiers sous DOS. Ce double emploi est dû aux difficultés actuelles rencontrées lors de la programmation sous Windows, notre objectif étant de n'avoir qu'une seule commande d'impression pour **MESSOR**.

La durée de stockage d'un message varie de zéro à l'infini ou presque et ne semble guère limitée que par la capacité physique du dispositif de mémorisation, il faudra de temps en temps faire de la place.

Ainsi un *fichier chrono* est crée sur le SUN pour pouvoir y sauver les "messages" reçus ou ceux envoyés. La commande *Sauver* du menu Services permet d'y faire des sauvegardes et les commandes *Consulter* et *Purger* du menu ***Chrono*** grâce à un montage d'une partition du SUN vont permettre de le consulter ou d'y effacer certains "messages".

Dans le mode courrier comme c'est souvent le cas au niveau de l'ORSTOM (il existe aussi le mode conférence ou forum), le stockage s'effectue sur une machine privilégiée, appelée serveur de messagerie qui au demeurant n'est pas dédiée à ce seul usage (nous l'avons vu un peu plus haut), mais qui doit être, en permanence, en service.

Pour la réception des "messages" une partition du SUN (exemple net use g: \dakar\home\spool\mail), pourra être montée par **MESSOR** pour pouvoir récupérer les **entêtes** des "messages" (qui seront affichés dés l'appel à **MESSOR** si il y a des messages) et leurs **contenus** pour les traiter localement.

Ainsi le système de messagerie **MESSOR** délivre une copie du "message" en différé dès qu'il peut le recevoir, assure à la demande l'envoi d'accusés de réception, alimente le fichier historique nommé ***chrono***, mais, sauf spécification contraire, ne supprime pas le "message" du serveur une fois que tous les destinataires l'ont reçu

Comme toute bonne messagerie **MESSOR** permettra d'associer des fichiers de n'importe quel type à l'envoie de "message".
Le "message" devient alors un commentaire par rapport au document envoyé. C'est pourquoi les options de commandes suivantes ont été retenues pour le traitement de fichiers :

- pour les petits fichiers en mode texte, une commande de transfert de type : *mail nom_de_fichier* comme sous UNIX dans le menu **Services**.

- pour les gros fichiers, les commandes **compression** et **décompression** du menu ***TRANSFORMATION*** permettront respectivement de réduire leur taille avant leur envoi et de les récupérer à la réception.

Pour faciliter la désignation des destinataires, **MESSOR** doit pouvoir disposer de l'annuaire du R.I.O.. C'est ainsi que le menu **Annuaire** permet de faire des consultations (commande **Consulter**) et des recherches (commande **Recherche**) sur l'annuaire électronique du R.I.O. grâce à une partition (net use s: \dakar\usr\local) du SUN qui sera monté au moment voulu par **MESSOR**.

XI - LES DIFFERENTES FONCTIONS MESSOR

Comme nous l'avons vu plus haut une application Window est spécialement écrite pour s'exécuter sous l'environnement Windows en utilisant le WAPI (Windows Application Program Interface). Elle possède deux bases fondamentales :

- une fonction principale nommée ***WinMain***
- les fonctions windows

La fonction ***WinMain*** est le point d'entrée d'une application et est similaire à la fonction main utilisée dans un environnement C standard. Elle est toujours nommée ainsi.

Une **fonction window** est quelque chose de nouveau. C'est une "*callback function*", une fonction à l'intérieur de notre application que Windows appelle.

Une application n'appelle jamais directement ses fonctions. Ainsi, elle attend Windows pour les appeler avec la requête d'exécuter une tâche spécifique ou de retourner une information.

Nous allons essayer de résumer l'essentiel du contenu de ces deux types de fonction.

XI.1 - La fonction WinMain ()

Comme la fonction ***main*** dans un programme C standard, toute application Windows doit intégrer, la fonction ***WinMain***. Aucune application ne peut s'exécuter sans cette fonction.

Dans la plupart des applications, la structure générale de cette fonction est la suivante :

- Appel des fonctions d'initialisation qui enregistrent les classes windows, créent des fenêtres ou qui effectuent tout autre initialisation nécessaire.

- Inscrit une boucle de gestion de la file d'attente des messages à partir la file d'attente des applications.

- Termine l'application quand la boucle de gestion des messages rencontre le message WM_QUIT.

La fonction ***WinMain*** a la forme suivante :

```
int PASCAL PWinMain (hInstance, hPreInstance,
                     lpCmdLine, nCmdShow)
HANDLE hInstance;
HANDLE hPreInstance;
LPSTR lpCmdLine;
int nCmdShow;
{

}
```

La fonction ***WinMain*** requière l'attribut PASCAL (voir Window.h).
Quand l'utilisateur démarre l'application, Windows communique les quatre paramètres suivants à la fonction WinMain de l'application.

Paramètres	valeurs que Windows passe à l'application
hInstance	une copie du numéro unique correspondant à l'application donnée.
hPrevInstance	une autre copie du numéro de l'application si une autre application s'exécute. Si il n'y a en pas ce paramètre est mis à NULL par Windows.
lpCmdLine	un pointer de type long qui pointe sur le caractère null d'une ligne de commande.
nCmdShow	une valeur entière qui spécifie si on doit afficher l'application sous forme de fenêtre ou sous forme d'icône.

Pour plus d'amples informations sur ces paramètres voir les manuels de référence 1 et 2 de Microsoft Windows.

XI.2 - Les fonctions "windows" de "MESSOR"

Chaque fenêtre doit disposer d'une fonction window. Cette fonction exécute les "messages" envoyés par le système de gestion des "messages" de Windows. Elle doit être définie dans la partie **EXPORTS** du fichier de définition (.DEF).

Elle a la forme suivante :

```
long FAR PASCAL Nom_de_la_fonction (hwnd, message, wParam,
                                    lParam)

HWND hwnd           /* numéro de la fonction , un entier */
unsigned message    /* type de message                   */
WORD wParam         /* information additionnelle         */
LONG lParam         /* information additionnelle         */

{
   .
   .
   .
    switch (message)
    {
      .
      .
      .
      default :
        return (DefWindowProc(hwnd,message,wParam,lParam);
    }
    return (NULL);
}
```

Les Fonctions utilisées ont pour mission d'ouvrir une large fenêtre, qui contiendra :

- le titre de l'application,
- la barre des menus,
- une petite fenêtre qui s'agrandit lorsqu'on clique dessus avec la souris permettant ainsi l'affichage des entêtes de messages si la boîte aux lettres n'est pas vide ou si elle est vide une fenêtre de dialogue informera l'utilisateur.

2.1 La fonction InitApplication ()

Cette fonction initialise les données correspondantes aux caractéristiques de la fenêtre "mère" et enregistre les différentes "classes" de celle-ci. Elle est comme la fonction **InitInstance** appelée à l'initialisation par la fonction **WinMain**. Parmi ces données, quatre reçoivent des valeurs importantes. Ce sont les champs qui permettent d'extraire les "messages" de cette classe, l'application à laquelle appartient cette classe, le nom du menu dans le fichier ressources (.RC), le nom utilisé lors de la création de la fenêtre "mère". Elle exécute ses tâches une seule fois et pointe dès l'initialisation sur la fonction **MainWndProc**.

2.2 La fonction InitInstance ()

Appelée à l'initialisation WinMain, elle retourne un booléen. Elle sauvegarde des copies de numéros de fenêtres dans des variables statiques qui peuvent être utilisées dans toute autre fenêtre, elle crée aussi la fenêtre "mère" avec la fonction prédéfinie *CreatWindow* et l'affiche par *ShowWindow*.

Ces deux fonctions permettent en fait de simplifier l'écriture de la fonction **WinMain** qui se partage ainsi en trois petites fonctions. De ce fait elles sont assez différentes des autres fonctions windows sur la manière de les écrire.

2.3 La fonction MainWndProc ()

Elle se divise en deux parties, la première, si elle existe comme dans le cas de **MESSOR**, permet au démarrage de l'application d'effectuer certaines tâches grâce à l'option **WM_CREATE(*)** (voir Manuel de référence 1). La deuxième, permet l'exécution des tâches demandées par l'utilisateur lorsque ce dernier effectue un choix dans un menu.

Dans le cas de **MESSOR**, la première partie nous permet d'afficher au démarrage l'état de la boîte à lettres de l'utilisateur. Si elle est vide un message est affiché, dans le cas contraire, les entêtes des "message" reçus sont affichées

grâce à un **ComboBox** (petite fenêtre) . L'utilisateur pourra en sélectionnant une entête lire le contenu du message correspondant.

La deuxième partie grâce au "message" **WM_COMMAND** traite le choix effectué par l'utilisateur dans le menu (*voir MESSOR.C*). Dans le fichier **MESSOR.RC**, à chaque champs du menu ou des fenêtres de dialogue correspond un mnémonique qui commence respectivement par IDM_ ou par ID. Ces identificateurs sont définis dans le fichier **MESSOR.H** avec un numéro unique.

Les identificateurs sont les différentes options du **WM_COMMAND**. C'est le paramètre **wParam** qui récupère le message correspondant au choix de l'utilisateur et demande à la fonction **MainWndProc** d'exécuter la tâche demandée.

La fonction **MainWndProc**, gère aussi entièrement le menu **Edition**, l'option **Imprimer** du menu **Message**, la fermeture, l'agrandissement d'une fenêtre et son déplacement.

MainWndProc permet d'exécuter toutes les fonctions citées ci-dessous.

2.4 Les fonctions appelées dans MainWndProc ()

Ces fonctions sont directement ou indirectement appelées par les choix existants dans la fonction MainWndProc (voir ci-dessus).

2.4.1 EnvoyerLet()

Cette fonction appelée par le choix de *nouveau message* du menu permet de récupérer les **nom**, **adresse** du **destinataire** et le **sujet** du "message" par l'intermédiaire de la fenêtre de dialogue *EnvoyetLetBox*. Elle ouvre par la suite une fenêtre d'édition pour la saisie du "message".

2.4.2 OpenFile ()

Elle ouvre une fenêtre (list box) contenant les noms des fichiers du répertoire courant. Le répertoire pouvant être changé par l'introduction d'un autre répertoire. Lorsque l'utilisateur effectue son choix, **OpenFile** opère comme ceci :

- elle récupère le nom du fichier cherche
- vérifie si l'ouverture est possible
  si oui elle examine la taille du fichier
  si la taille n'excéde pas celle du buffer d'édition
  alors elle affiche le contenu dans la fenêtre d'édition
  sinon elle envoie un message d'erreur
  sinon message d'erreur
- sinon message d'erreur

2.4.3 SaveAsDlg ()

Cette fonction permet d'effectuer des sauvegardes de fichier. Elle fait appel à des fonctions telles que ***CheckFileName, SeparateFileName()***,(voir ci-dessus), avant d'effectuer la sauvegarde dans le répertoire spécifié ou par défaut.

2.4.4 TransfertFichier ()

Elle permet le transfert direct de fichiers par "mail" d'UNIX. Elle récupère par la fenêtre de dialogue **TransfertBox** du fichier (.DLG) le nom du fichier puis le destinataire et le sujet qui seront ajoutés comme entête au fichier. Une vérification de l'existence du destinataire est faite grâce à la fonction **RechercheAdresse ()**

2.4.5 DelivrerMessage ()

Elle récupère après que l'utilisateur ait fini de taper son "message", le(s) nom(s) et adresse de(s) destinataire(s) pour l'ajouter comme entête au "message". Elle vérifie grâce à la fonction **RechercheAdresse ()** si le destinataire existe si oui le message est délivré sinon un message d'erreur est envoyé.

2.4.6 EffacerMessage ()

Permet de marquer un "message" par le caractère # qu'on place devant son nom empêchant ainsi l'accès à ce dernier. Si à la fin de la session le caractère existe encore le "message" correspondant sera effacé définitivement.

2.4.7 RecupererMessage ()

Récupère les "messages" effacés en enlevant le caractère # devant le nom du "message", permettant ainsi l'accès à celui-ci.

2.4.8 SauverMessage ()

Elle permet de sauver les "messages" reçus dans le fichier **Chrono** qui se trouve dans une partition du disque du Serveur que *SauverMessage* montera à chaque appel.

2.4.9 ImprimerMessage ()

Elle permet d'imprimer les "messages" reçu et qui n'ont pas encore été transférés dans le Chrono.

2.4.10 ConsulterChrono ()

Récupère le fichier chrono en montant une partition du Serveur et affiche dans le **ComboBox** crée dans le **WM_CREATE** les "messages" qui y ont été sauvés.

2.4.11 PurgerChrono ()

Elle permet de se positionner sur la partition du Serveur contenant le fichier **Chrono** pour y effacer des "messages" de manière définitive.

2.4.12 ConsulterAnnuaire ()

Elle monte une partition de Serveur contenant le fichier annuaire, transforme ce dernier en fichier temporaire DOS puis fait appel à la fenêtre d'édition pour l'afficher.

2.4.13 RechercheAdresse ()

Recherche une adresse, retourne FAUX si le nom spécifié n'est pas dans l'annuaire et VRAI sinon. Un message est envoyé pour chaque cas.

2.4.14 AbortDlg ()

Elle affiche le nom du fichier à imprimer et offre par la même occasion à l'utilisateur, la possibilité d'arrêter l'impression. Elle est appelée lors du choix de l'option IDM_PRINT de MainWndProc.

2.4.15 About(), OKMessageBox ()

Ces fonctions, lorsqu'elles sont appelées, permettent d'envoyer des messages à la fenêtre qui à fait l'appel. Elles permettent surtout d'envoyer des messages d'erreurs ou pour confirmer des commandes.

2.4.16 GetPrinterDc ()

C'est une fonction qui entre en communication avec le fichier **WIN.INI** de Windows 3 pour y extraire les caractéristiques de l'imprimante en vigueur et le port de sortie.

2.4.17 AbortProg ()

Elle examine s'il y a des "messages" dans la file d'attente d'impression et permet d'annuler les demandes d'impression.

2.4.18 SetNewBuffer ()

Elle permet d'allouer un nouveau buffer d'édition en libérant la place occupée par le buffer actuelle et en redessinant l'écran.

2.4.19 QuerySaveFile ()

Elle s'assure si le fichier courant a été modifié ou non, si tel est le cas elle demande à l'utilisateur s'il veut sauvegarder les changements intervenus. En cas de réponse favorable, elle appelle la fonction **SaveAsDlg** pour faire la sauvegarde.

2.4.20 SaveFile ()

Elle ouvre le fichier en écriture avec l'option **OF_CREATE** qui écrase le contenu précédent par le nouveau contenu du buffer d'édition.

2.4.21 CheckFileName ()

La fonction vérifie, si un nom de fichier est présent dans le cas d'un rappel, si elle ne contient pas de caractères interdits dans le cas ou on donne un nom à un fichier. Elle demandera à l'utilisateur s'il doit l'écraser dans le cas d'une sauvegarde.

2.4.22 SeparateFile ()

Divise le chemin absolue en deux parties et les copie dans deux variables différentes l'une étant le nom du fichier l'autre le reste du chemin d'accès.

2.4.23 ChangeDefExt () et AddExt ()

Permettent respectivement de changer l'extension ou d'ajouter l'extension par défaut à un nom de fichier.

2.4.24 UpdateListBox ()

Elle construit un chemin absolue d'accès à un répertoire ou à un fichier en concaténant le nom au chemin relatif. Elle récupère les données pour les afficher dans la fenêtre pour la fenêtre appelée : List Box.

2.4.25 ConnexionRes ()

Cette fonction permet à un autre réseau de se connecter. Elle est prévue pour des développements futurs.

Les options des menus Transformation et Aide n'ont pas encore été traitées dans l'application **MESSOR** pour la simple raison que la programmation n'est pas encore terminée.

Le détail de toutes ces fonctions est présenté dans les cinq (5) fichiers **MESSOR** ***(MESSOR.DLG, MESSOR.RC, MESSOR.H, MESSOR.DEF, MESSOR.C).*** Mais pour comprendre le fonctionnement global de l'application, il est encore nécessaire d'intégrer une aide au programme compilé. Cette partie est actuellement en construction et pourra bientôt être ajoutée à **MESSOR**.

TROISIEME PARTIE

PROGRAMME M E S S O R

MESSOR.RC,
MESSOR.DLG
MESSOR.H.,
MESSOR.C.,
MESSOR.DEF

MESSOR.RC

```
#include "windows.h"
#include "Messor.h"


MessorFileMenu MENU
BEGIN
        POPUP       "&Message"
        BEGIN
        MENUITEM  "&Nouveau Message",       IDM_NEW
        MENUITEM  "&Ouvrir sous dos",       IDM_OPEN
        MENUITEM  SEPARATOR
        MENUITEM  "&Enregistrer",           IDM_SAVE
        MENUITEM  "Enregistrer &Sous",      IDM_SAVEAS
        MENUITEM  SEPARATOR
        MENUITEM  "&Imprimer",              IDM_PRINT
        MENUITEM  SEPARATOR
        MENUITEM  "&Fermer...",             IDM_FERMER
        MENUITEM  "&Quitter MESSOR" ,       IDM_EXIT
        MENUITEM  SEPARATOR
        MENUITEM  "&A propos de MESSOR",  IDM_ABOUT
    END


    POPUP     "&Edition"
    BEGIN
      MENUITEM  "&Annuler\tAlt+BkSp",       IDM_UNDO
      MENUITEM  SEPARATOR
      MENUITEM  "C&ouper\tShift+Del", IDM_CUT
      MENUITEM  "&Copier\tCtrl+Ins",        IDM_COPY
      MENUITEM  "Co&ller\tShift+Ins",   IDM_PASTE
      MENUITEM  "&Nettoyer\tDel",           IDM_CLEAR
    END

    POPUP     "&Services"
    BEGIN
      MENUITEM    "&Tranf. de Fichier",  IDM_TRANSFERT
      MENUITEM    SEPARATOR
      MENUITEM    "&Délivrer"                 IDM_DELIVRER
      MENUITEM    "&Accusé",                  IDM_ACCUSE
      MENUITEM    SEPARATOR
      MENUITEM    "&Effacer",                 IDM_EFFACER
      MENUITEM    "&Récupèrer",               IDM_RECUPERE
      MENUITEM    SEPARATOR
      MENUITEM    "&Sauver",                  IDM_SAUVER
      MENUITEM    "&Imprimer",                IDM_IMPRIME
    END


    POPUP     "&Chrono"
    BEGIN
      MENUITEM    "&Consulter",               IDM_CONSULTE
      MENUITEM    SEPARATOR
      MENUITEM    "&Purger...",               IDM_PURGE
    END
```

```
  POPUP      "&Transformation"
  BEGIN
   MENUITEM    "&Encodage...",              IDM_CODER
   MENUITEM    "&Désencodage",                    IDM_DECODER
   MENUITEM    SEPARATOR
   MENUITEM    "Com&pression",                    IDM_COMPRESS
   MENUITEM    "Dé&compression",            IDM_DECOMPRESS
  END


  POPUP     "Ann&uaire"
  BEGIN
   MENUITEM  "&Consulter",                  IDM_CONSULTER
   MENUITEM  SEPARATOR
   MENUITEM  "&Recherche",                  IDM_RECHERCHE
  END


  POPUP      "\a&Aide"
  BEGIN
   MENUITEM    "&Aide....",                 IDM_AIDE
   MENUITEM    SEPARATOR
   MENUITEM    "A &Propos de Aide",         IDM_HELP
  END

END


ServicesMenu MENU
BEGIN
  MENUITEM  "&Répondre",                          IDM_REPLY
  POPUP     "&Montrer "
   BEGIN
    MENUITEM  "&Suivant",                         IDM_NEXT
    MENUITEM  "&Précedent",                 IDM_BEFORE
   END
END


MessorFileAcc ACCELERATORS
BEGIN
  VK_BACK,  IDM_UNDO,  VIRTKEY, ALT
  VK_DELETE, IDM_CUT,  VIRTKEY, SHIFT
  VK_INSERT, IDM_COPY,  VIRTKEY, CONTROL
  VK_INSERT, IDM_PASTE, VIRTKEY, SHIFT
  VK_DELETE, IDM_CLEAR, VIRTKEY,       SHIFT
END



#include "Messorst.dlg"
```

MESSOR.DLG

```
Open DIALOG  40, 40, 180, 112
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "Ouvrir "
BEGIN
  LTEXT "Ouvrir le Fichier de &Nom :", IDC_FILENAME, 4,
                                    , 100, 10
  EDITTEXT              IDC_EDIT,    4, 16, 100,
                    12, ES_AUTOHSCROLL
      LTEXT "&Fichiers  dans :",IDC_FILES,   4, 32 ,100, 10
  LISTBOX,              IDC_LISTBOX,  4, 52,  70, 56,
                    WS_TABSTOP | WS_VSCROLL
  LTEXT "",             IDC_PATH,    90, 32, 100, 10
  DEFPUSHBUTTON "&Ouvrir", IDOK,       87, 60,  50, 14
  PUSHBUTTON "Annuler",    IDCANCEL,   87, 80,  50, 14
END


Saveas DIALOG 40, 40, 180, 53
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "Enregistrer  "
BEGIN
  LTEXT "Enregistrer sous le &nom :", IDC_FILENAME,  4, 4,
                                                  100, 10
  LTEXT "",             IDC_PATH,    90, 4, 92,   10
  EDITTEXT              IDC_EDIT,     4, 16, 100, 12
  DEFPUSHBUTTON  "Sauver",       IDOK,   120, 16, 50,  14
  PUSHBUTTON    "Annuler",   IDCANCEL,   120, 36, 50,  14
END


Sortie DIALOG 40, 40, 148, 112
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "SORTIE"
BEGIN
  LTEXT "    Lancer la  Sortie  sur ", IDC_FILENAME, 15,
                                  10, 100, 12
  LTEXT "        PERIPHERIQUE",    IDC_PATH,  16, 35,
                                        160, 12
  DEFPUSHBUTTON  "Imprimante",       IDC_PRINT, 4, 50, 50,
                                                    12,
  PUSHBUTTON    "Ecran",         IDC_ECRAN, 80, 50, 50,
                                        12, WS_GROUP
  PUSHBUTTON    "Lancer",           IDOK, 4, 80, 40,
                                  14,
  PUSHBUTTON    "Annuler"          IDCANCEL, 80, 80, 40,
                                  14, WS_GROUP
END
```

```
AboutBox DIALOG 42, 47, 144, 122
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "A Propos de MESSAGERIE"
BEGIN
   CTEXT "Microsoft Windows"              -1,       5, 5,144,8
   CTEXT "Application Messagerie - ORSTOM" -1,      0,14,144,8
   CTEXT "par  Mr PAPA  ABDOU  DIALLO "    -1,      0,23,144,8
   CTEXT "élève ingénieur  à l'E.N.S.U.T." -1,      0,30,144,8
   CTEXT "Sous la direction de"           -1, 0,37,144,8
   CTEXT "Mr Hervé CHEVILLOTTE."          -1,0,44,144,8
   CTEXT "O.R.S.T.O.M. - DAKAR HANN."      -1,      0,67,144,8
   CTEXT " ANNEE 1991"                    -1,       10,80,144,8
   DEFPUSHBUTTON "OK"          IDOK,     53, 100, 32, 14,
                                          WS_GROUP
END


AboutHelp DIALOG 72, 77, 144, 75
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "A Propos de l'Aide"
BEGIN
   CTEXT "Permet  de  consulter "  -1,      0,  5, 144, 8
   CTEXT "l'Annuaire  des  adresses" -1,      0, 14, 144,8
   CTEXT "et toutes les fonctionnalitˇs",-1,    0,23,144,8
   CTEXT "de la MESSAGERIE  O.R.S.T.O.M." -1,   0,32,144, 14
   DEFPUSHBUTTON  "OK"            IDOK,     53,59,32,14,
                                          WS_GROUP
END


AbortDlg DIALOG 20,20,90,64
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "MESSAGERIE - ORSTOM"
BEGIN
 DEFPushButton "Annuler"  IDCANCEL, 29,44,32,14, WS_GROUP
 Ctext "Envoyer",        -1,       0, 8,90,8
 Ctext "texte",       IDC_FILENAME,  0,18,90,8
 Ctext "à la file d'impression", -1, 0,28,90,8
END
```

```
EnvoyerBox DIALOG 50, 60, 220, 120
STYLE WS_POPUP | DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "Adressage puis composition de message"
BEGIN
LTEXT "&Destinataire :",     IDC_EDIT,     4, 4, 100,
                              10,WS_GROUP
EDITTEXT                    IDC_DESTINATION, 60,4, 140, 12
                             ,ES_AUTOHSCROLL
LTEXT  "&Objet      :",     IDC_OBJLET  4, 34, 100, 10
EDITTEXT                    IDC_OBJET, 60, 34, 140, 12
                             ,ES_AUTOHSCROLL
LTEXT "&Copie A     :",     IDC_CC,    4, 64, 100, 10
EDITTEXT                    IDC_TRACE, 60, 65, 140, 12
                            ,ES_AUTOHSCROLL
PUSHBUTTON "&Composer",      IDOK,     25,  95, 52, 12 ,
                             WS_GROUP
 PUSHBUTTON "&Annuler",      IDCANCEL, 125, 95, 52, 14
                            END

ConnexionBox DIALOG  10, 10, 220, 112
STYLE WS_POPUP | DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "CONNEXION AU RESEAU"
BEGIN
LTEXT "Nom de la &connexion :",  IDC_CONNEXION,    2, 5, 100,
                                  12, WS_GROUP
EDITTEXT                    IDC_HOST,      80, 5 100, 12
LTEXT "Nom de &login        :", IDC_LOGIN,    2, 35, 100, 12
EDITTEXT                    IDC_USER       80, 35, 100,12
LTEXT "&Mot de passe        :", IDC_MOTDEPASS, 2, 65, 100, 12
EDITTEXT                    IDC_PASSWD,   80, 65, 100,12,
                                  ES_PASSWORD
PUSHBUTTON "&Connexion",         IDOK,         25, 90, 52, 14
                                  , WS_GROUP
PUSHBUTTON "&Annuler",           IDCANCEL      125, 90, 52, 14
END



BoiteaLettre DIALOG 40, 70, 200, 122
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU| DS_MODALFRAME |
            WS_MINIMIZEBOX | WS_MAXIMIZEBOX
CAPTION "Affichage des Messages recu"
MENU ServicesMenu
BEGIN
EDITTEXT             IDD_BUFFER,        3,3, 190, 110,
                    ES_MULTILINE | WS_VSCROLL
END
```

MESSOR.H

```
/* ce fichier est le fichier des définitions et des                    déclarations*/


/* champs du menu fichiers */


#define   IDM_NEW      100
#define   IDM_OPEN     101
#define   IDM_SAVE     102
#define   IDM_SAVEAS    103
#define   IDM_PRINT    105
#define   IDM_FERMER    106
#define   IDM_EXIT     107
#define   IDM_ABOUT     108


/* champs du menu edition */


#define   IDM_UNDO     200
#define   IDM_CUT      201
#define   IDM_COPY     202
#define   IDM_PASTE    203
#define   IDM_CLEAR     204


/* champs du menu services */


#define   IDM_TRANSFERT 300
#define   IDM_DELIVRER  301
#define   IDM_ACCUSE    302
#define   IDM_EFFACER   303
#define   IDM_RECUPERE  304
#define   IDM_SAUVER    305
#define   IDM_IMPRIME   306


/* champs du menu chrono */


#define   IDM_CONSULTE  400
#define   IDM_PURGE     401
```

```
/* champs du menu transformation */

#define   IDM_CODER     500
#define   IDM_DECODER   501
#define         IDM_COMPRESS   502
#define   IDM_DECOMPRESS 503


/* champs du menu annuaire */

#define   IDM_CONSULTER 900
#define   IDM_RECHERCHE 901

                :

/* champs du menu Aide */

#define   IDM_AIDE      600
#define   IDM_CONSULT   601
#define   IDM_HELP      602


/* Control IDs */

#define   IDC_FILENAME    700
#define   IDC_EDIT        701
#define   IDC_FILES       702
#define   IDC_PATH        703
#define   IDC_LISTBOX     704
#define   IDC_DESTINATION 705
#define   IDC_OBJET       706
#define   IDC_TRACE       707
#define   IDC_CONNEXION   708
#define   IDC_USER        709
#define   IDC_MOTDEPASS   710
#define   IDC_PRINT       716
#define   IDC_ECRAN       717
#define   IDC_COMBOBOX    712
#define   IDC_DESTINE     711
#define   IDC_OBJLET      718
#define   IDC_CC          719
#define   IDC_COMPOSER    720
#define   IDC_PASSWD      721
#define   IDC_HOST        722
#define   IDC_LOGIN       723
```

```
/* champs du bouton montrer */


#define    IDM_COMBOBOX  801
#define    ID_COMBOBOX   803


/* option du menu boxlettre */


#define    IDM_REPLY    903
#define    IDM_NEXT     904
#define    IDM_BEFORE   905
#define    IDD_BUFFER   906


#define MAXFILESIZE 0x7fff        /* maximum file size that
                                     can be loaded */


/* déclaration des procédures à utiliser */


int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);
long FAR PASCAL MainWndProc(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL About(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL ComboBoxExemple(HWND, unsigned, WORD, LONG);
HANDLE FAR PASCAL OpenDlg(HWND, unsigned, WORD, LONG);
int FAR PASCAL SaveAsDlg(HWND, unsigned, WORD, LONG);
BOOL CheckFileName(HWND, PSTR, PSTR);
BOOL SaveFile(HWND);
BOOL QuerySaveFile(HWND);
void SeparateFile(HWND, LPSTR, LPSTR, LPSTR);
void UpdateListBox(HWND);
void SetNewBuffer(HWND, HANDLE, PSTR);
void AddExt(PSTR, PSTR);
void ChangeDefExt(PSTR, PSTR);
HANDLE GetPrinterDC();
int FAR PASCAL AbortProc(HDC, int);
int FAR PASCAL AbortDlg(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL ConnexionRes(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL EnvoyerLet(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL Lettre(HWND, unsigned, WORD, LONG);
```

```
/*************************************************************/

   PROGRAM: Messor.c

   PURPOSE: Loads, saves, and edits text files

   FUNCTIONS:

      WinMain() - calls initialization function, processes message loop
      InitApplication() - initializes window data and
           registers window
      InitInstance() - saves instance handle and creates
         main window
      MainWndProc() - processes messages
      About() - processes messages for "About" dialog box
      SaveAsDlg() - save file under different name
      OpenDlg() - let user select a file, and open it.
      UpdateListBox() - Update the list box of OpenDlg
      ChangeDefExt() - Change the default extension
      SeparateFile() - Separate filename and pathname
      AddExt() - Add default extension
      CheckFileName() - Check for wildcards, add extension
          if needed
      SaveFile() - Save current file
      QuerySaveFile() - Called when some action might lose
          current contents
      SetNewBuffer() - Set new buffer for edit window

*************************************************************/


#include "windows.h"
#include "prntfile.h"
#include "io.h"
#include "stdio.h"
#include "stdlib.h"
#include "process.h"


HANDLE hInst;
HANDLE hAccTable;           /* handle to accelerator table */
HWND hEditWnd;              /* handle to edit window */
HWND hCmboWnd;
HWND hwnd;                  /* handle to main window */
```

```
/* Additional includes needed for the fstat() function */


#include <sys\types.h>
#include <sys\stat.h>


char FileName[128];
char PathName[128];
char OpenName[128];
char DefPath[128];
char DefSpec[13] = "*.*";
char DefExt[] = ".msg";
char str[255];
char FichierEntete[] = "g:entete.msg";


HANDLE hEditBuffer;                    /* handle to editing buffer   */
HANDLE hOldBuffer;                     /* old buffer handle          */
HANDLE hHourGlass;                     /* handle to hourglass cursor */
HANDLE hSaveCursor;                    /* current cursor handle      */


int hFile;                          /* file handle                   */
int count;                          /* number of chars read or
                                           written */
PSTR pBuffer;                               /* address of read/write buffer
                                */
OFSTRUCT OfStruct;                          /* information from OpenFile() */
struct stat FileStatus;          /* information from fstat()     */
BOOL bChanges = FALSE;           /* TRUE if the file is changed */
BOOL bSaveEnabled = FALSE;       /* TRUE if text in the edit
                                           buffer   */
PSTR pEditBuffer;                /* address of the edit buffer  */
RECT Rect;                       /* dimension of the client window
                */


WORD n;
char *szBuff;
char *StrBuffer;


char AppName [] = "Application :  Messagerie O.R.S.T.O.M
                                P.A.DIALLO";


char Untitled[] =                    /* default window
                                title     */
                " Composition et ˇdition des messages P.A.DIALLO ";
```

```
                    /* Printer variables */

HDC hPr;                    /* handle for printer  device context */
int LineSpace;      /* spacing between lines                      */
int LinesPerPage;   /* lines per page                                 */
int CurrentLine;    /* current line                               */
int LineLength;     /* line length                                    */
DWORD dwLines;              /* number of lines to print               */
DWORD dwIndex;              /* index into lines to print              */
char pLine[128];    /* buffer to store lines before printing */
TEXTMETRIC TextMetric; /* information about character size */
BOOL bAbort;                /* FALSE if user cancels printing         */
HWND hAbortDlgWnd;

FARPROC lpAbortDlg, lpAbortProc;
```

```
/*************************************************************

        FUNCTION: WinMain(HANDLE, HANDLE, LPSTR, int)

        PURPOSE: calls initialization function, processes message
loop

*************************************************************/


int PASCAL WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
HANDLE hInstance;
HANDLE hPrevInstance;
LPSTR lpCmdLine;
int nCmdShow;
{
        MSG msg;

        if (!hPrevInstance)
                if (!InitApplication(hInstance))
                  return (FALSE);

        if (!InitInstance(hInstance, nCmdShow))
                return (FALSE);

        while (GetMessage(&msg, NULL, NULL, NULL))
        {

  /* Only translate message if it is not an accelerator message */

     if (!TranslateAccelerator(hwnd, hAccTable, &msg))
        {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
       }
  }
  return (msg.wParam);
}
```

```
/*****************************************************************

  FONCTION : AskConfirmation (HWND)

    OBJET : Ask confirmation for exit

*****************************************************************/

AskConfirmation (HWND hwnd)

  {
   return MessageBox (hwnd, "Voulez-vous r˘ellement quitter
l'application?",
                              AppName, MB_YESNO | MB_ICONQUESTION);
  }

/*****************************************************************/

void OkMessageBox(HWND hwnd, char *szString, char *szFileName)
    {
     char szBuffer [50];

     wsprintf(szBuffer, szString, (LPSTR) szFileName);
     }

/*****************************************************************/


/*****************************************************************

        FUNCTION: InitApplication(HANDLE)

        PURPOSE: Initializes window data and registers window
class

*****************************************************************/

BOOL InitApplication(hInstance)
HANDLE hInstance;
{
   WNDCLASS  wc;

   wc.style = NULL;
   wc.lpfnWndProc = MainWndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInstance;
   wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
```

```
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName =  "MessorFileMenu";
    wc.lpszClassName = "MessorFileWClass";

    return (RegisterClass(&wc));
}


/*******************************************************************

    FUNCTION: InitInstance(HANDLE, int)

    PURPOSE: Saves instance handle and creates main window

*******************************************************************/


BOOL InitInstance(hInstance, nCmdShow)
    HANDLE          hInstance;
    int             nCmdShow;
{
    RECT            Rect;
    hInst = hInstance;

    hAccTable = LoadAccelerators(hInst, "MessorFileAcc");

    hwnd = CreateWindow(
                        "MessorFileWClass",
                "   Application :  MESSAGERIE - O.R.S.T.O.M. P. A.
        DIALLO",
                        WS_OVERLAPPEDWINDOW ,
                        0,
                        0,
                        640,
                        480,
                        NULL,
                        NULL,
                        hInstance,
                        NULL);

        if (!hwnd)
        return (FALSE);

        ShowWindow(hwnd, nCmdShow);
        UpdateWindow(hwnd);
}
```

```
/*****************************************************************

        FUNCTION: MainWndProc(HWND, unsigned, WORD, LONG)

        PURPOSE:  Processes messages

        MESSAGES:

        WM_COMMAND    - application menu (About dialog box)
        WM_DESTROY    - destroy window
        WM_
        WM_SIZE       - window size has changed
        WM_QUERYENDSESSION - willing to end session?
        WM_ENDSESSION - end Windows session
        WM_CLOSE      - close the window
        WM_SIZE       - window resized

        COMMENTS:



*****************************************************************/


long FAR PASCAL MainWndProc(hWnd, message, wParam, lParam)
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;

{
   FARPROC lpProcAbout,  lpOpenDlg, lpSaveAsDlg,
lpEnvoyerLet, lpConnexionRes, lpLettre;


   int Success;          /* return value from SaveAsDlg() */
   int IOStatus;                 /* result of file i/o     */
   int nPageSize;                /* vert. resolution of printer device
        */
   LONG lSelect;
   WORD wEnable;
   RECT Rect;
   HDC hdc;
   TEXTMETRIC tm;
   int nIndex;
   int i, j ;
   HANDLE nFiles;
   static int cxChar,cyChar;
   long int taille;
```

```
switch (message)
{  case WM_CREATE :

      GetClientRect (hWnd, (LPRECT) &Rect);

      hCmboWnd = CreateWindow ("Combobox",
                                    NULL,
                                    WS_CHILD | WS_VISIBLE |
                                    WS_VSCROLL | CBS_SORT |
                                    ES_AUTOVSCROLL|
                                    WS_HSCROLL | CBS_AUTOHSCROLL |
                                    CBS_HASSTRINGS | CBS_DROPDOWNLIST,
                                    2,
                                    2,
                                    630,
                                    100,
                                    hWnd,
                                    IDC_COMBOBOX,
                                    hInst,
                                    NULL);

            if (!hCmboWnd)
            {
                  DestroyWindow (hWnd);
            return (NULL);
            }


        hFile = OpenFile("c:\\entete", &OfStruct, OF_READ);
        if (hFile >= 0)
            {
             taille = filelength (hFile);
             szBuff = malloc(taille);
             StrBuffer = malloc(taille);
             nIndex = read(hFile, szBuff, taille);
             close (hFile);
             j = 0;
             for (i = 0; i <= taille; i++)
             {
               :
     StrBuffer[j] = szBuff[i];
              if (StrBuffer[j] == 0x0D)
     {
       StrBuffer[j] = 0x20;
                StrBuffer[j+1] = 0x20;
                StrBuffer[0] = 0x20;
            SendMessage(hCmboWnd, CB_INSERTSTRING, 0x0000,
                          (LONG)(LPSTR) StrBuffer);
                j = 0;
       free(StrBuffer);
                StrBuffer = malloc(taille);

              }
              else j++;
             }
```

```
            }
 else
            {
             MessageBox(hwnd, "Fichier introuvable.", NULL,
                    MB_OK | MB_ICONHAND);
             return (NULL);
        }

  break;

 case WM_COMMAND:
    switch (wParam)
       {
       case IDM_ABOUT:
                  lpProcAbout = MakeProcInstance(About,
                                                 hInst);
              DialogBox(hInst, "AboutBox", hWnd,
             lpProcAbout);
          FreeProcInstance(lpProcAbout);
          break;


     /* case CBN_SELCHANGE:
           n = (WORD) SendMessage(hCmboWnd,
   CB_GETCURSEL, 0, 0L);
                n = (WORD) SendMessage(hCmboWnd, B_GETLBTEXT,
                    n,(LONG) (LPSTR) szBuffer);
                strcpy (szBuffer + n + 1, getenv (szBuffer) );
                *(szBuffer + n) = '=';
                SetWindowText(hEditWnd, szBuffer);
                return 0;*/


       case IDM_NEW:

/*  If current file has been modified, query user about
                  * saving it. */

         . if (!QuerySaveFile(hWnd))
            return (NULL);

     /* bChanges is set to FALSE to indicate there have been
* no changes since the last file save. */

          bChanges = FALSE;
          FileName[0] = 0;

         /* Update the edit buffer */
          if (hEditWnd)
            SetNewBuffer(hWnd, NULL, Untitled);
          else
                        {
```

```
            lpEnvoyerLet =
              MakeProcInstance(EnvoyerLet, hInst);
            DialogBox(hInst, "EnvoyerBox", hWnd,
                    lpEnvoyerLet);



            FreeProcInstance(lpEnvoyerLet);
                }
            break;


        case IDM_OPEN:
            if (!QuerySaveFile(hWnd))
              return (NULL);

            lpOpenDlg = MakeProcInstance((FARPROC)
                                  OpenDlg, hInst);


            /* Open the file and get its handle */
            hFile = DialogBox(hInst, "Open", hWnd,
              lpOpenDlg);
            FreeProcInstance(lpOpenDlg);
            if (!hFile)
              return (NULL);

    /* Allocate edit buffer to the size of the file + 1 */


            hEditBuffer =
              LocalAlloc(LMEM_MOVEABLE |
LMEM_ZEROINIT,(WORD)FileStatus.st_size+1);

            if (!hEditBuffer)
                {
              MessageBox(hWnd, "Pas assez de
              memoire.",
                                  NULL, MB_OK | MB_ICONHAND);
              return (NULL);
            }
            hSaveCursor = SetCursor(hHourGlass);
            pEditBuffer = LocalLock(hEditBuffer);

            IOStatus = read(hFile, pEditBuffer,
              FileStatus.st_size);
            close(hFile);

            /* # bytes read must equal file size */

            if (IOStatus != (int)FileStatus.st_size) {

              sprintf(str, "Erreur de lecture %s.",
        FileName);
```

```
            SetCursor(hSaveCursor);     /* Remove the hourglass */
            MessageBox(hWnd, str, NULL, MB_OK |
            MB_ICONEXCLAMATION);
          }

          LocalUnlock(hEditBuffer);

          /* Set up a new buffer and window title */

          sprintf(str, "Application : Messagerie -
                        ORSTOM - %s",
                                        FileName);
          SetNewBuffer(hWnd, hEditBuffer, str);
          SetCursor(hSaveCursor);       /* restore the cursor */
          break;


        case IDM_SAVE:

 /* If there is no filename, use the saveas command to get
       * one.  Otherwise, save the file using the current filename.*/


          if (!FileName[0])
            goto saveas;
          if (bChanges)
            SaveFile(hWnd);
          break;


        case IDM_SAVEAS:
saveas:
          lpSaveAsDlg = MakeProcInstance(SaveAsDlg,
                    hInst);

          /* Call the SaveAsDlg() function to get
the new filename */

          Success = DialogBox(hInst, "SaveAs", hWnd,
            lpSaveAsDlg);
          FreeProcInstance(lpSaveAsDlg);

 /* If successful, update the window title, save the file */

          if (Success == IDOK)
              {
            sprintf(str, "Messagerie ORSTOM -
                        %s", FileName);
            SetWindowText(hWnd, str);
            SaveFile(hWnd);
           }
          break;
```

```
                                        /* User canceled */
    case IDM_PRINT:
      hSaveCursor = SetCursor(hHourGlass);
      hPr = GetPrinterDC();
      if (!hPr)
            {
        sprintf(str, "Ne peut pas imprimer
        %s", FileName);
        MessageBox(hWnd, str, NULL, MB_OK |
        MB_ICONHAND);
        return (NULL);
       }

      lpAbortDlg =  MakeProcInstance(AbortDlg,
                                        hInst);
      lpAbortProc = MakeProcInstance(AbortProc,
                                    hInst);

      /* Define the abort function */

      Escape(hPr, SETABORTPROC, NULL,
        (LPSTR) (long) lpAbortProc,
        (LPSTR) NULL);

      if (Escape(hPr, STARTDOC, 4, "PrntFile
        text", (LPSTR) NULL) < 0)
            {
      MessageBox(hWnd,"Incapable de commencer
l'impression",
                                    NULL, MB_OK | MB_ICONHAND);
        FreeProcInstance(lpAbortDlg);
        FreeProcInstance(lpAbortProc);
        DeleteDC(hPr);
      }

    bAbort = FALSE; /* Clears the abort flag  */


    /* Create the Abort dialog box (modeless) */


      hAbortDlgWnd = CreateDialog(hInst,
"AbortDlg",
                                        hWnd, lpAbortDlg);

                        if (!hAbortDlgWnd)
          {
                SetCursor(hSaveCursor);     /* Remove the
                                        hourglass */
        MessageBox(hWnd, "NULL Abort window
                handle",
            NULL, MB_OK | MB_ICONHAND);
        return (FALSE);
       }
```

```
            /* Now show Abort dialog */

            ShowWindow (hAbortDlgWnd, SW_NORMAL);

            /* Disable the main window to avoid
reentrancy problems */

            EnableWindow(hWnd, FALSE);
            SetCursor(hSaveCursor);     /* Remove the hourglass */
```

/* Since you may have more than one line, you need to compute the spacing between lines. You can do that by retrieving the height of the characters you are printing and advancing their height plus the recommended external leading height. */

```
            GetTextMetrics(hPr, &TextMetric);
            LineSpace = TextMetric.tmHeight +
              TextMetric.tmExternalLeading;
```

/* Since you may have more lines than can fit on one page, you need to compute the number of lines you can * print per page. You can do that by retrieving the * dimensions of the page and dividing the height by the line spacing. */

```
                nPageSize = GetDeviceCaps (hPr, VERTRES);
                LinesPerPage = nPageSize / LineSpace - 1;
```

/* You can output only one line at a time, so you need a count of the number of lines to print. You can retrieve the count sending the EM_GETLINECOUNT message to the edit control. */

```
        dwLines = SendMessage(hEditWnd,
    EM_GETLINECOUNT, 0, 0L);
```

```
 /* Keep track of the current line on the current page */

          CurrentLine = 1;

/* One way to output one line at a time is to retrieve
 * one line at a time from the edit control and write it
 * using the TextOut function.  For each line you need to
 * advance one line space.  Also, you need to check for the
 * end of the page and start a new page if necessary.
*/

          for (dwIndex = IOStatus = 0; dwIndex <
            dwLines; dwIndex++) {
            pLine[0] = 128;
       /* maximun buffer size */
            pLine[1] = 0;
            LineLength = SendMessage(hEditWnd,
     EM_GETLINE,
            (WORD)dwIndex, (LONG)((LPSTR)pLine));
            TextOut(hPr, 0, CurrentLine*LineSpace,
              (LPSTR)pLine, LineLength);
            if (++CurrentLine > LinesPerPage )
                    {
              CurrentLine = 1;
              IOStatus = Escape(hPr, NEWFRAME,
                    0, 0L, 0L);
              if (IOStatus<0 || bAbort)
                 break;
             }
          }

          if (IOStatus >= 0 && !bAbort)
                {
            Escape(hPr, NEWFRAME, 0, 0L, 0L);
            Escape(hPr, ENDDOC, 0, 0L, 0L);
           }
          EnableWindow(hWnd, TRUE);

          /* Destroy the Abort dialog box */

          DestroyWindow(hAbortDlgWnd);
          FreeProcInstance(lpAbortDlg);
          FreeProcInstance(lpAbortProc);
          DeleteDC(hPr);
          break;
```

```
case IDM_FERMER :
        if (!QuerySaveFile(hWnd))
            return (NULL);

    bChanges = FALSE;
        FileName[0] = 0;
    SetNewBuffer(hEditWnd, NULL, NULL);
        DestroyWindow(hEditWnd);
        hEditWnd = 0;
    SetNewBuffer(hWnd, NULL, AppName);
        break;

/* edit menu commands */

case IDM_UNDO:
        SendMessage (hEditWnd, WM_UNDO, 0, 0L);
   return 0;

case IDM_CUT:
        SendMessage (hEditWnd, WM_CUT, 0, 0L);
   return 0;

case IDM_COPY:
        SendMessage (hEditWnd, WM_COPY, 0, 0L);
   return 0;

case IDM_PASTE:
        SendMessage (hEditWnd, WM_PASTE, 0, 0L);
   return 0;

case IDM_CLEAR:
        SendMessage (hEditWnd, WM_CLEAR, 0, 0L);
        return 0;

break;

/* Commandes de l'option Fonctions */


case IDM_TRANSFERT :
         case IDM_DELIVRER  :
    MessageBox (
       GetFocus(),
      "APPLICATION MESSOR"
      "Private ",
     MB_ICONASTERISK | MB_OK);
    break;



     case IDM_ACCUSE   :
     case IDM_EFFACER  :
     case IDM_RECUPERE :
     case IDM_SAUVER   :
     case IDM_IMPRIME  :
```

```
        MessageBox (
        GetFocus(),
      "Application MESSAGERIE ORSTOM.",
      "PRIVATE",
       MB_ICONASTERISK | MB_OK);
    break;

 /* Commande de l'option Chrono */

          case IDM_CONSULTE :
          case IDM_PURGE    :
        MessageBox (
        GetFocus(),
      "Application MESSAGERIE ORSTOM",
      "PRIVATE",
       MB_ICONASTERISK | MB_OK);
        break;


/* Commandes de l'option Transformation */
          case IDM_CODER    :
          case IDM_DECODER  :
          case IDM_COMPRESS :
          case IDM_DECOMPRESS :
             MessageBox (
        GetFocus(),
      "Application MESSAGERIE ORSTOM",
      "PRIVATE",
         MB_ICONASTERISK | MB_OK);
       break;

/* option du menu annuaire */

          case IDM_CONSULTER:
          case IDM_RECHERCHE:
             MessageBox ( GetFocus(),
        "MESSAGERIE ORSTOM",
                        "PRIVATE",
                        MB_ICONASTERISK | MB_OK);
                        break;

/* Commandes de l'option Aide */

    case IDM_AIDE :
            MessageBox (
            GetFocus (),
          "Commande non encore impl'ment'e.",
          "Application MESSAGERIE ORSTOM.",
             MB_ICONASTERISK | MB_OK);
             break;
```

```
        case IDM_HELP :
         lpProcAbout = MakeProcInstance(About,
                  hInst);
         DialogBox(hInst, "AboutHelp", hWnd,
   lpProcAbout);
         FreeProcInstance(lpProcAbout);
         break;


      case IDM_EXIT:
         QuerySaveFile(hWnd);
              if (IDYES == AskConfirmation (hwnd))
           DestroyWindow(hWnd);
         break;


      case IDC_EDIT:
         if (HIWORD (lParam) == EN_ERRSPACE)
               {
           MessageBox (
               GetFocus ()
             , "Depassement de la m_moire."
             , "Application : Messagerie
                  O.R.S.T.O.M   P.A.D."
             , MB_ICONHAND | MB_OK
           );
           }
         break;

   }
   break;


case WM_SETFOCUS:
   SetFocus (hEditWnd);
   break;


case WM_SIZE:
     MoveWindow(hEditWnd, 2 ,200, 635, 220,TRUE);
   break;


case WM_INITMENUPOPUP :
      if (lParam == 1)
        {
      EnableMenuItem (wParam, IDM_UNDO,
            SendMessage (hEditWnd, EM_CANUNDO, 0, 0L) ?
                 MF_ENABLED : MF_GRAYED);

      EnableMenuItem (wParam, IDM_PASTE,
                 IsClipboardFormatAvailable (CF_TEXT) ?
                  MF_ENABLED : MF_GRAYED);

          lSelect = SendMessage (hEditWnd, EM_GETSEL, 0, 0L);
```

```
            if (HIWORD (lSelect) == LOWORD (lSelect))
              wEnable = MF_GRAYED ;
            else
              wEnable = MF_ENABLED ;

          EnableMenuItem (wParam, IDM_CUT, wEnable);
          EnableMenuItem (wParam, IDM_COPY, wEnable);
                    EnableMenuItem (wParam, IDM_CLEAR, wEnable);
            return 0;
          }
        break;


    case WM_QUERYENDSESSION:
              /* message: to end the session? */
        if (IDYES == AskConfirmation (hwnd))
            return 1L;
        else
            return 0;


    case WM_CLOSE:     /* message: close the window    */
       if (IDYES == AskConfirmation (hWnd))
         DestroyWindow(hWnd);
       return 0;


    case WM_DESTROY:
       PostQuitMessage(0);
       break;


    default:
       return (DefWindowProc(hWnd, message, wParam,
                             lParam));
  }
  return (NULL);
}


/****************************************************************

  FUNCTION: SaveAsDlg(HWND, unsigned, WORD, LONG)

  PURPOSE: Allows user to change name to save file to

  COMMENTS:

    This will initialize the window class if it is the first time this application is run. It then
creates the window, and processes the message loop until a PostQuitMessage is received. It
exits the application by returning the value passed by the PostQuitMessage.

****************************************************************/
```

```
int FAR PASCAL SaveAsDlg(hDlg, message, wParam, lParam)
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
   char TempName[128];

   switch (message)
    {
      case WM_INITDIALOG:

       /* If no filename is entered, don't allow the user to save to it */

        if (!FileName[0])
           bSaveEnabled = FALSE;
        else {
           bSaveEnabled = TRUE;

           /* Process the path to fit within the IDC_PATH field */

           DlgDirList(hDlg, DefPath, NULL, IDC_PATH,
                                              0x4010);

       /* Send the current filename to the edit control */

           SetDlgItemText(hDlg, IDC_EDIT, FileName);

          /* Accept all characters in the edit control */

          SendDlgItemMessage(hDlg, IDC_EDIT, EM_SETSEL, 0,
              MAKELONG(0, 0x7fff));
        }

        /* Enable or disable the save control depending on whether the filename exists.*/


        EnableWindow(GetDlgItem(hDlg, IDOK),
bSaveEnabled);

      /* Set the focus to the edit control within the dialog
                                  box */

        SetFocus(GetDlgItem(hDlg, IDC_EDIT));
        return (FALSE);              /* FALSE since
                        Focus was changed */


      case WM_COMMAND:
        switch (wParam)
           {
```

```
        case IDC_EDIT:

      /* If there was previously no filename in the edit
       * control, then the save control must be enabled as
       * soon as a character is entered. */


          if (HIWORD(lParam) == EN_CHANGE &&
     !bSaveEnabled)
          EnableWindow(GetDlgItem(hDlg, IDOK),
     bSaveEnabled = TRUE);
          return (TRUE);


        case IDOK:

         /* Get the filename from the edit control */

          GetDlgItemText(hDlg, IDC_EDIT, TempName,
                                        128);

       /* If there are no wildcards, then separate the name
     *into path and name. If a path was specified, replace
     *the default path with the new path.*/


          if (CheckFileName(hDlg, FileName,
                              TempName))
                {
            SeparateFile(hDlg, (LPSTR) str,
     (LPSTR) DefSpec,
                (LPSTR) FileName);
            if (str[0])
              strcpy(DefPath, str);

         /* Tell the caller a filename was selected */

              EndDialog(hDlg, IDOK);
          }
          return (TRUE);

        case IDCANCEL:

          /* Tell the caller the user canceled the
            SaveAs function */

          EndDialog(hDlg, IDCANCEL);
          return (TRUE);
      }
      break;
  }


  return (FALSE);
}
```

```
/*************************************************************

  FUNCTION: OpenDlg(HWND, unsigned, WORD, LONG)

  PURPOSE: Let user select a file, and open it.

*************************************************************/


HANDLE FAR PASCAL OpenDlg(hDlg, message, wParam, lParam)
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
  WORD index;
  PSTR pTptr;
  HANDLE hFile;

  switch (message)
   {
     case WM_COMMAND:
        switch (wParam)
           {

          case IDC_LISTBOX:
             switch (HIWORD(lParam))
                   {

                case LBN_SELCHANGE:
          /* If item is a directory name, append "*.*" */


                        if (DlgDirSelect(hDlg, str, IDC_LISTBOX))
                        {
                   strcat(str, DefSpec);
                        }
                   SetDlgItemText(hDlg, IDC_EDIT,
                                  str);
                   SendDlgItemMessage(hDlg,
                      IDC_EDIT,
                      EM_SETSEL,
                      NULL,
                      MAKELONG(0, 0x7fff));
                   break;


                case LBN_DBLCLK:
                   goto openfile;
             }
             return (TRUE);
```

```
        case IDOK:
openfile:
            GetDlgItemText(hDlg, IDC_EDIT, OpenName,
                                    128);
            if (strchr(OpenName, '*') ||
              strchr(OpenName, '?'))
                {
              SeparateFile(hDlg, (LPSTR) str,
                    (LPSTR) DefSpec,
                (LPSTR) OpenName);
              if (str[0])
                strcpy(DefPath, str);
              ChangeDefExt(DefExt, DefSpec);
              UpdateListBox(hDlg);
              return (TRUE);
            }

            if (!OpenName[0])
                {
              MessageBox(hDlg, "Nom de fichier non
                    specifie.",
                NULL, MB_OK | MB_ICONHAND);
              return (TRUE);
            }

            AddExt(OpenName, DefExt);

            /* Open the file */

            if ((hFile = OpenFile(OpenName,
              (LPOFSTRUCT) &OfStruct,
                OF_READ)) == -1)
                {
              sprintf(str, "Erreur %d  a l'ouverture
                            %s.",
                OfStruct.nErrCode, OpenName);
              MessageBox(hDlg, str, NULL,
                MB_OK | MB_ICONHAND);
            }
            else
                            {

              /* Make sure there's enough room for
      the file */

              fstat(hFile, &FileStatus);

              if (FileStatus.st_size > MAXFILESIZE)
{
                sprintf(str,
            "Pas assez de memoire pour charger %s.\n%s
supérieur à %ld bits.",
                  OpenName, OpenName,
                    MAXFILESIZE);
```

```
                    MessageBox(hDlg, str, NULL,
                       MB_OK | MB_ICONHAND);
                    return (TRUE);
                  }

                  /* File is opened and there is enough
room so return
                   * the handle to the caller.
                   */

                  strcpy(FileName, OpenName);
                  EndDialog(hDlg, hFile);
                  return (TRUE);
                }
               return (TRUE);

            case IDCANCEL:
                  /* strcpy(DefPath, str);
                   ChangeDefExt(DefExt, DefSpec);*/
               EndDialog(hDlg, NULL);
               return (TRUE);
          }
          break;

       case WM_INITDIALOG:          /* message: initialize */

          UpdateListBox(hDlg);
          SetDlgItemText(hDlg, IDC_EDIT, DefSpec);
          SendDlgItemMessage(hDlg,            /* dialog
                                handle     */
            IDC_EDIT,                      /* where to
          send message  */
            EM_SETSEL,                      /* select
                        characters     */
            NULL,                          /*
                 additional information */
            MAKELONG(0, 0x7fff));           /* entire
                                contents     */
          SetFocus(GetDlgItem(hDlg, IDC_EDIT));
          return (FALSE); /* Indicates the focus is set to a
          control */
                  }
    return FALSE;
}

/****************************************************************

    FUNCTION: UpdateListBox(HWND);

    PURPOSE: Update the list box of OpenDlg

****************************************************************/
```

```
void UpdateListBox(hDlg)
HWND hDlg;
{
   strcpy(str, DefPath);
   strcat(str, DefSpec);
   DlgDirList(hDlg, str, IDC_LISTBOX, IDC_PATH, 0x4010);

   /* To ensure that the listing is made for a subdir. of
    * current drive dir...
    */
   if (!strchr (DefPath, ':'))
        DlgDirList(hDlg, DefSpec, IDC_LISTBOX, IDC_PATH, 0x4010);

   /* Remove the '..' character from path if it exists, since this will make DlgDirList move us
up an additional level in the tree when UpdateListBox() is called again. */

   if (strstr (DefPath, ".."))
        DefPath[0] = '\0';
   SetDlgItemText(hDlg, IDC_EDIT, DefSpec);
}


/*****************************************************************

   FUNCTION: ChangeDefExt(PSTR, PSTR);

   PURPOSE: Change the default extension

*****************************************************************/


void ChangeDefExt(Ext, Name)
PSTR Ext, Name;
{
   PSTR pTptr;
   pTptr = Name;
   while (*pTptr && *pTptr != '.')

      pTptr++;
   if (*pTptr)
      if (!strchr(pTptr, '*') && !strchr(pTptr, '?'))
         strcpy(Ext, pTptr);
}


/*****************************************************************

   FUNCTION: SeparateFile(HWND, LPSTR, LPSTR, LPSTR)

   PURPOSE: Separate filename and pathname

*****************************************************************/
```

```
void SeparateFile(hDlg, lpDestPath, lpDestFileName, lpSrcFileName)
HWND hDlg;
LPSTR lpDestPath, lpDestFileName, lpSrcFileName;
{
   LPSTR lpTmp;
   char  cTmp;

   lpTmp = lpSrcFileName + (long) lstrlen(lpSrcFileName);
   while (*lpTmp != ':' && *lpTmp != '\\' && lpTmp > lpSrcFileName)
      lpTmp = AnsiPrev(lpSrcFileName, lpTmp);
   if (*lpTmp != ':' && *lpTmp != '\\')
   {
      lstrcpy(lpDestFileName, lpSrcFileName);
      lpDestPath[0] = 0;
      return;
   }
   lstrcpy(lpDestFileName, lpTmp + 1);
   cTmp = *(lpTmp + 1);
   lstrcpy(lpDestPath, lpSrcFileName);
    *(lpTmp + 1) = cTmp;
   lpDestPath[(lpTmp - lpSrcFileName) + 1] = 0;
}


/*****************************************************************

   FUNCTION: AddExt(PSTR, PSTR);

   PURPOSE: Add default extension

*****************************************************************/


void AddExt(Name, Ext)
PSTR Name, Ext;
{
   PSTR pTptr;
   pTptr = Name;
   while (*pTptr && *pTptr != '.')
      pTptr++;
   if (*pTptr != '.')
      strcat(Name, Ext);
}
```

```
/*****************************************************************

   FUNCTION: CheckFileName(HWND, PSTR, PSTR)

   PURPOSE: Check for wildcards, add extension if needed

   COMMENTS:

   Make sure you have a filename and that it does not contain any wildcards. If needed, add
the default extension. This function is called whenever your application wants to save a file.

*****************************************************************/


BOOL CheckFileName(hWnd, pDest, pSrc)
HWND hWnd;
PSTR pDest, pSrc;
{
   PSTR pTmp;

   if (!pSrc[0])
      return (FALSE);            /* Indicates no filename
                was specified */

   pTmp = pSrc;
   while (*pTmp)                 /* Searches the string
                        for wildcards */
   {
      switch (*pTmp++)
         {
         case '*':
         case '?':
            MessageBox(hWnd, "Caract.re joker non
                  permis.",
               NULL, MB_OK | MB_ICONEXCLAMATION);
            return (FALSE);
      }
   }

   AddExt(pSrc, DefExt);         /* Adds the default
                              extension if needed */

   if (OpenFile(pSrc, (LPOFSTRUCT) &OfStruct, OF_EXIST) >= 0)
   {
      sprintf(str, "Remplacer le fichier existant %s?",
pSrc);
      if (MessageBox(hWnd, str, "PrntFile",
            MB_OKCANCEL | MB_ICONEXCLAMATION) == IDCANCEL)
         return (FALSE);
   }
   strcpy(pDest, pSrc);
   return (TRUE);
}
```

```
/*************************************************************

   FUNCTION: SaveFile(HWND)

   PURPOSE: Save current file

   COMMENTS:

   This saves the current contents of the Edit buffer, and  changes bChanges to indicate that
the buffer has not been changed since the last save.

  Before the edit buffer is sent, you must get its handle and lock it to get its address. Once the
file is written, you must unlock the buffer. This allows Windows to move the buffer when not
in immediate use.

*************************************************************/


BOOL SaveFile(hWnd)
HWND hWnd;
{
   BOOL bSuccess;
   int IOStatus;                        /* result
                        of a file write */

   if ((hFile = OpenFile(FileName, &OfStruct,
      OF_PROMPT | OF_CANCEL | OF_CREATE)) < 0)
   {

      /* If the file can't be saved */

      sprintf(str, "Ecriture impossible sur %s.", FileName);
      MessageBox(hWnd, str, NULL, MB_OK | MB_ICONHAND);
      return (FALSE);
   }


   hEditBuffer = SendMessage(hEditWnd, EM_GETHANDLE, 0, 0L);
   pEditBuffer = LocalLock(hEditBuffer);

   /* Set the cursor to an hourglass during the file transfer          */

   hSaveCursor = SetCursor(hHourGlass);
   lOStatus = write(hFile, pEditBuffer, strlen(pEditBuffer));
   close(hFile);
   SetCursor(hSaveCursor);

   if (IOStatus != strlen(pEditBuffer))
   {
      sprintf(str, "Erreurr ~ l'~criture sur %s.",
                                    FileName);
```

```
        MessageBox(hWnd, str,
            NULL, MB_OK | MB_ICONHAND);
        bSuccess = FALSE;
    }
    else
    {
        bSuccess = TRUE;        /* Indicates the file as saved */
        bChanges = FALSE;       /* Indicates changes have been saved */
    }

    LocalUnlock(hEditBuffer);
    return (bSuccess);
}


/****************************************************************

    FUNCTION: QuerySaveFile(HWND);

    PURPOSE: Called when some action might lose current contents

    COMMENTS:

    This function is called whenever we are about to take an action that would lose the current
contents of the edit buffer.

****************************************************************/


BOOL QuerySaveFile(hWnd)
HWND hWnd;
{
    int Response;
    FARPROC lpSaveAsDlg;

    if (bChanges)
    {
        sprintf(str, "Sauver les changements: %s", FileName);
        Response = MessageBox(hWnd, str,
            "PrntFile",  MB_YESNOCANCEL | MB_ICONHAND);
        if (Response == IDYES)
            {
check_name:

            /* Make sure there is a filename to save to */

            if (!FileName[0])
                {
                lpSaveAsDlg = MakeProcInstance(SaveAsDlg, hInst);
```

```
            Response = DialogBox(hInst, "SaveAs",
               hWnd, lpSaveAsDlg);
            FreeProcInstance(lpSaveAsDlg);
            if (Response == IDOK)
               goto check_name;
            else
               return (FALSE);
         }
         SaveFile(hWnd);
      }
      else if (Response == IDCANCEL)
         return (FALSE);
   }
   else
      return (TRUE);
}

/*************************************************************

   FUNCTION: SetNewBuffer(HWND, HANDLE, PSTR)

   PURPOSE: Set new buffer for edit window

   COMMENTS:

      Point the edit window to the new buffer, update the window title, and
      redraw the edit window.  If hNewBuffer is NULL, then create an empty
      1K buffer, and return its handle.

*************************************************************/


void SetNewBuffer(hWnd, hNewBuffer, Title)
HWND hWnd;
HANDLE hNewBuffer;
PSTR Title;
{
   HANDLE hOldBuffer;

   hOldBuffer = SendMessage(hEditWnd, EM_GETHANDLE, 0, 0L);
   LocalFree(hOldBuffer);
   if (!hNewBuffer)              /* Allocates a buffer
                                 if none exists */
      hNewBuffer = LocalAlloc(LMEM_MOVEABLE | LMEM_ZEROINIT,
                                 1);

   SendMessage(hEditWnd, EM_SETHANDLE, hNewBuffer, 0L); /*
                                 Updates the buffer
                                                   and displays new buffer
                                                                  */
   SetWindowText(hWnd, Title);
   SetFocus(hEditWnd);
```

```
    bChanges = FALSE;
}

/*****************************************************************

    FUNCTION: GetPrinterDC()

    PURPOSE:  Get hDc for current device on current output port according to
              info in WIN.INI.

    COMMENTS:

        Searches WIN.INI for information about what printer is connected, and
        if found, creates a DC for the printer.

        returns
            hDC > 0 if success
            hDC = 0 if failure

*****************************************************************/

HANDLE GetPrinterDC()
{
    char pPrintInfo[80];
    LPSTR lpTemp;
    LPSTR lpPrintType;
    LPSTR lpPrintDriver;
    LPSTR lpPrintPort;

    if (!GetProfileString("windows", "Device", (LPSTR)"",
                pPrintInfo, 80))
        return (NULL);
    lpTemp = lpPrintType = pPrintInfo;
    lpPrintDriver = lpPrintPort = 0;
    while (*lpTemp)
    {
        if (*lpTemp == ',')
            {
            *lpTemp++ = 0;
            while (*lpTemp == ' ')
                lpTemp = AnsiNext(lpTemp);
            if (!lpPrintDriver)
                lpPrintDriver = lpTemp;
            else
                {
                lpPrintPort = lpTemp;
                break;
                }
        }
        else
            lpTemp = AnsiNext(lpTemp);
```

```
    }

    return (CreateDC(lpPrintDriver, lpPrintType, lpPrintPort,                    (LPS
TR) NULL));
}


/************************************************************

    FUNCTION: AbortProc()

    PURPOSE:  Processes messages for the Abort Dialog box

************************************************************/


int FAR PASCAL AbortProc(hPr, Code)
HDC hPr;                    /* for multiple printer
                display contexts */
int Code;                   /* printing status  */
{
    MSG msg;

    /* Process messages intended for the abort dialog box */

    while (!bAbort && PeekMessage(&msg, NULL, NULL, NULL,
TRUE))
        if (!IsDialogMessage(hAbortDlgWnd, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }

    /* bAbort is TRUE (return is FALSE) if the user has                    aborted */

    return (!bAbort);
}


/************************************************************

    FUNCTION: AbortDlg(HWND, unsigned, WORD, LONG)

    PURPOSE:  Processes messages for printer abort dialog box

    MESSAGES:


        WM_INITDIALOG - initialize dialog box
        WM_COMMAND    - Input received
    COMMENTS
 This dialog box is created while the program is printing, and allows the user to cancel the
printing process.
************************************************************/
```

```
int FAR PASCAL AbortDlg(hDlg, msg, wParam, lParam)
HWND hDlg;
unsigned msg;
WORD wParam;
LONG lParam;
{
  switch(msg)
   {

    /* Watch for Cancel button, RETURN key, ESCAPE key, or                SPACE
BAR */

    case WM_COMMAND:
      return (bAbort = TRUE);

    case WM_INITDIALOG:

      /* Set the focus to the Cancel box of the dialog                    */

      SetFocus(GetDlgItem(hDlg, IDCANCEL));
      SetDlgItemText(hDlg, IDC_FILENAME, FileName);
      return (TRUE);
    }
  return (FALSE);
}


/*****************************************************************

  FUNCTION: About(HWND, unsigned, WORD, LONG)

  PURPOSE:  Processes messages for "About" dialog box

  MESSAGES:

    WM_INITDIALOG - initialize dialog box
    WM_COMMAND    - Input received

*****************************************************************/


BOOL FAR PASCAL About(hDlg, message, wParam, lParam)
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
  switch (message)
   {
    case WM_INITDIALOG:
      return (TRUE);

    case WM_COMMAND:
        if (wParam == IDOK
```

```
            || wParam == IDCANCEL)
            {
            EndDialog(hDlg, TRUE);
            return (TRUE);
          }
          return (TRUE);
    }
    return (FALSE);
}

/*************************************************************/

BOOL FAR PASCAL EnvoyerLet(hDlg, message, wParam, lParam)
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
   switch (message)
        {
         case WM_COMMAND :
            switch (wParam)
                {
                  case IDOK :
                GetClientRect(hwnd, (LPRECT) &Rect);

        /* Creation d'une fenetre fille de type editeur */

              hEditWnd = CreateWindow("Edit",
                    NULL,
                        WS_OVERLAPPEDWINDOW |
                        WS_CHILD | WS_VISIBLE |
                        ES_MULTILINE |
                      WS_VSCROLL |
                        WS_BORDER |
                        ES_AUTOVSCROLL,
                        2,
                        200,
                        635,
                      220,


                        hwnd,
                        IDC_EDIT,  /*    Child control i.d.
                                                          */
                        hInst,
                      NULL);

              if (!hEditWnd)
                {
                  DestroyWindow(hwnd);
                  return (NULL);
                }
```

```
                hHourGlass = LoadCursor (NULL, IDC_WAIT);

               if (!QuerySaveFile(hwnd))
                    return (NULL);
             bChanges = FALSE;
                    FileName[0] = 0;

               SetNewBuffer(hwnd, NULL, Untitled);
               break;

             }
             case IDCANCEL :
             EndDialog(hDlg, NULL);
             return (TRUE);
             break;
             default : return(FALSE);
    }

 }

/*****************************************************************

  FUNCTION: ConnexionRes(HWND, unsigned, WORD, LONG)

  PURPOSE:  Processes messages for "ConnexionBox" dialog box

  MESSAGES:

    WM_INITDIALOG - initialize dialog box
    WM_COMMAND    - Input received

*****************************************************************/

BOOL FAR PASCAL ConnexionRes(hDlg, message, wParam, lParam)
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
  switch (message)
   {
    case WM_COMMAND:
          switch (wParam)
           {
            case IDOK    : break;
          case IDCANCEL :
              EndDialog(hDlg, TRUE);
              return (TRUE);
                   break;

               case IDC_HOST : GetDlgItem(hDlg, IDC_CONNEXION);
               case IDC_USER : GetDlgItem(hDlg, IDC_LOGIN);
               case IDC_PASSWD : GetDlgItem(hDlg, IDC_MOTDEPASS);
                              break;
```

```
        }
    }
        return (FALSE);
}

/*************************************************************/

BOOL FAR PASCAL Lettre(hDlg, message, wParam, lParam)
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
   switch (message)
    {
      case WM_INITDIALOG:
          return (TRUE);

            case WM_COMMAND:
            case  IDM_REPLY :
            case  IDM_NEXT :
            case  IDM_BEFORE :
            case  IDOK : break;
            case  WM_SIZE : /*MoveWindow(hEditWnd, 2 ,200, 635,
                                                    220,TRUE);*/
            case IDCANCEL :
            EndDialog(hDlg, TRUE);
            return (TRUE);
   }
   return (FALSE);
}
```

MESSOR.DEF

```
NAME      Messor

DESCRIPTION  'Application MESSAGERIE O.R.S.T.O.M   PAR PAPA
       ABDOU DIALLO'

EXETYPE          WINDOWS

STUB             'WINSTUB.EXE'

CODE             PRELOAD MOVEABLE DISCARDABLE
DATA             PRELOAD MOVEABLE MULTIPLE

HEAPSIZE         0xAFFF   ;   45k
STACKSIZE        8192

/*  cette partie décrit les fonctions exportée c'est àdire non windows*/

EXPORTS
                MainWndProc      @1
                About            @2
                OpenDlg                  @3
                SaveAsDlg                @4    ; appelée pour le menu
                                               ; sauver item selecté
                AbortDlg                 @5    ; pour annuler l'impression
                AbortProc                @6    ; traite les messages
                ConnexionRes     @7    ; connexion au reseau
                EnvoyerLet       @8    ; envoie de lettre a un ou
                                       ; plusieurs destinataires
                Lettre           @9
                TransfertFichier @10
                DelivrerMessage  @11
                EffacerMessage   @12
                ConsulterChrono  @13
                ConsulterAnnuaire @14
                RecupererMessage @15
```

CONCLUSION

Ce cas concret de développement de la messagerie UNIX sous une interface assurant l'utilisation des menus déroulants, du multi-fenêtrage et des protocoles de communication du système UNIX, avait pour ambition d'une part de montrer le détail de la programmation des différentes modules nécessaires à la mise en place d'un produit fiable et d'autre part de présenter l'important travail d'études nécessaires à sa réalisation.

Comme nous l'avons vu, les différentes commandes de MESSOR permettent d'envoyer un message à un nom, un nom de groupe, de diffuser un message général (à plusieurs destinataires) ou de transférer des fichiers par la messagerie. Ces commandes permettent également la réception d'un message en provenance d'un nom, d'un nom de groupe ou de qui que ce soit sur le Réseau Informatique Orstom.

Bien que l'application MESSOR soit immédiatement utilisable, elle reste cependant d'un niveau modeste au regard à ce qui est encore possible de faire si nous arrivons à surmonter tous les problèmes que posent l'exploitation du logiciel Windows, qui est tout à fait nouveau. La communication avec le DOS n'étant pas très évidente.

Cependant, elle nous a permis de mettre en valeur :

- les différentes techniques utilisées pour la transmission des messages (données),

- la situation d'une application messagerie par rapport au modèle OSI et ses différentes interactions avec les différentes couches OSI,

- les multiples fonctionnalités d'une messagerie,

- l'intérêt d'une bonne documentation et la manière de s'en servir,

- la pertinence et l'intérêt pour les pays en voie de développement, (cas concret d'une coopération utile à tous les égards entre l'ISRA et l'ORSTOM au Sénégal), de se doter de réseaux locaux d'ordinateurs avec en toile de fond la communication inter et intra-réseaux,

- l'enjeu des réseaux de communications qui dépassent les avantages concurrentiels liés à l'accroissance de la productivité, aux gains économiques à l'amélioration des produits et services...

Des perspectives telles que l'implémentation de **MESSOR**, dans un futur réseau reliant l'Université de Saint-Louis, de Dakar et l'ENSUT, ne poserait à priori aucun problème. Car actuellement, notre travail est centré sur l'intégration de tous types de réseau.

En dernière analyse, le moule de l'application MESSOR est parfaitement définie, il suffit de le réutiliser pour toute autre application. La souplesse et ses automatismes poussés permettent d'obtenir des présentations excellentes en s'affranchissant des tâches "ingrates" relaives à la gestion des affichages et de l'impression.

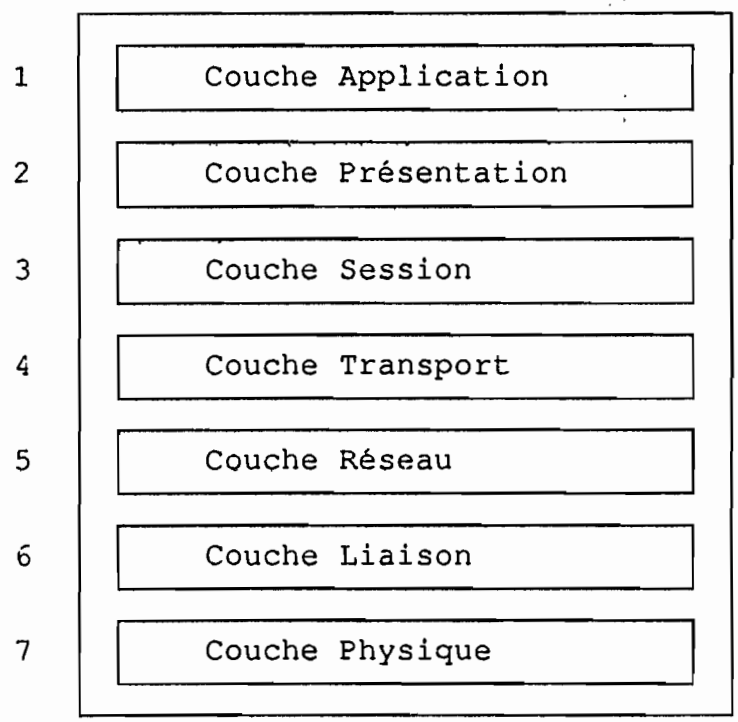
Dans cet optique, tout programmeur pourra utiliser ce moule pour de futurs développements. Nous sommes persuadés qu'il découvrira mille façons d'utiliser judicieusement les outils qui lui sont présentés.

A N N E X E 1

LE MODELE DE REFERENCE ISO

Le besoin généralisé de pouvoir interconnecter des équipements informatiques hétérogènes, a conduit l'ISO à définir une norme d'architecture de réseau informatique au sein de laquelle puissent prendre place des protocoles normalisés permettant la communication entre réseaux hétérogènes.

L'architecture OSI (Open System Interconnection) est constituée de sept couches selon le schéma suivant :



Les Couches OSI

LES FONCTIONS DES COUCHES OSI :

- La couche physique contrôle le circuit de données. Sur ce niveau l'unité d'information est le bit.

- La couche liaison correspond au contrôle logique de la liaison de données selon un protocole défini. L'information à ce niveau est la trame.

- La couche réseau gère le multiplexage des paquets et leur acheminement. L'information à ce niveau est le paquet.

- La couche transport a pour fonction le réassemblage des paquets en message. L'unité d'information pour cette couche est le message.

- La couche session réalise la mise en place et le contrôle du dialogue entre tâches ou transactions telles que l'activation d'une commande, la synchronisation.

- La couche présentation contrôle l'ensemble des informations ou données échangées entre utilisateurs. Elle assure la gestion des entrées et des sorties

- La couche application permet l'exécution des commandes liées aux processus de l'application.

Il faut noter que tout réseau informatique ne respecte pas obligatoirement le modèle à 7 couches de l'OSI.

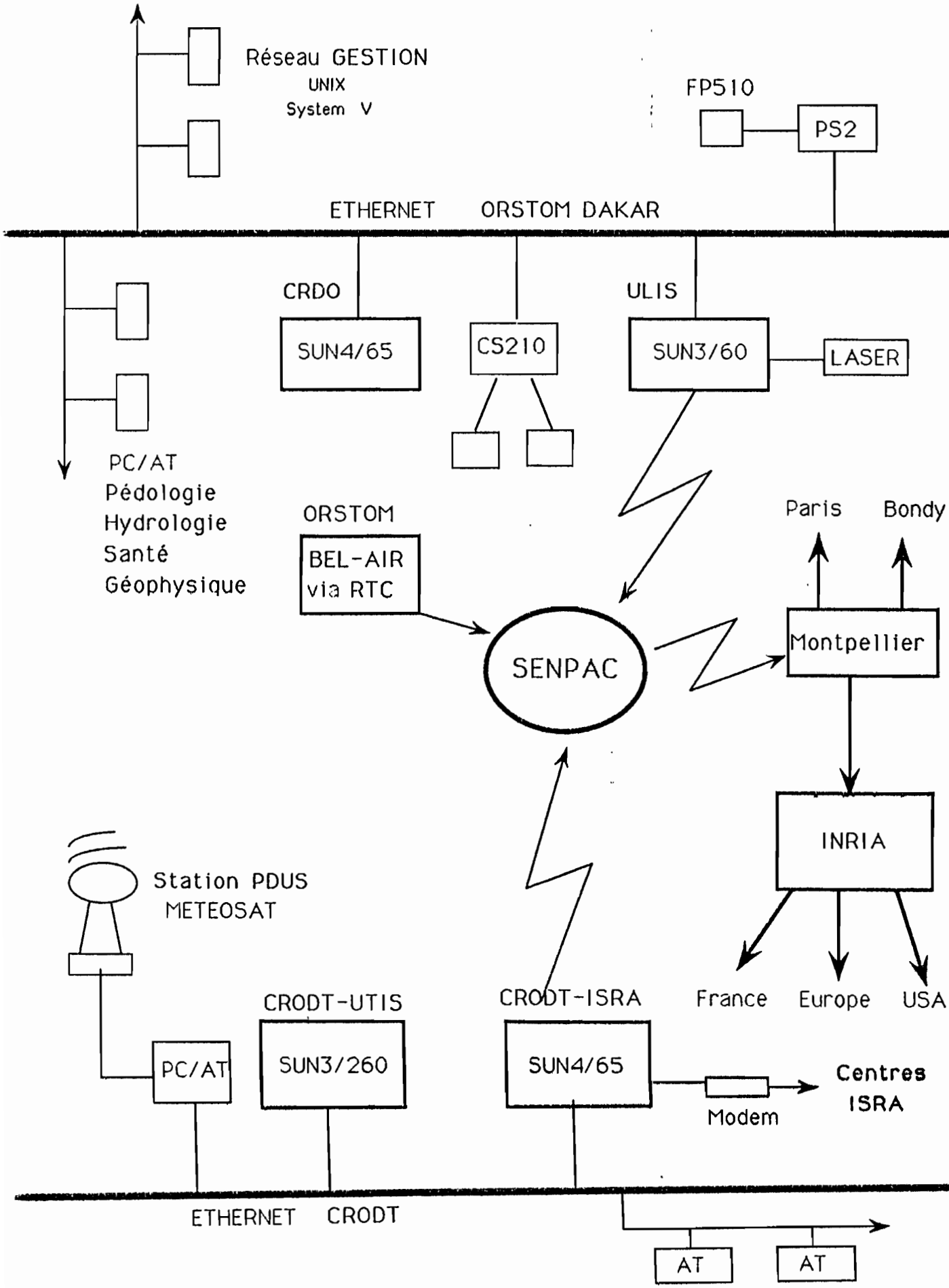
Dans un réseau constitué par des ETTD homogènes, les trois premières couches sont suffisantes pour dialoguer entre elles. Par contre dans le cas d'un réseau auquel sont connectés des ETTD hétérogènes les 7 couches OSI sont nécessaires.

LES NOEUDS DE LIAISON DU RESEAU INFORMATIQUES DE L'ORSTOM A TRAVERS 7 PAYS CIBLES

IMPLANTATION EQUIPEE ET ADRESSE	TYPE DE SERVEUR RESEAU	LIAISONS TELECOM	DEBIT DE LA LIGNE	PROTOCOLES	PERIODICITE DE LA LIAISON
Montpellier orstom.orstom.fr	SUN 3/60	Transpac : 8 CV RTC : 2 lignes 1: 1200 bps 2: 2400 MNP5	9600 bps	IP/X25 UUCP/F UUCP/G	immediat avec INRIA Rocquencourt
Bondy bondy.orstom.fr	SUN 3/260	Transpac 8 CV	9600 bps	IP/X25 UUCP/F	immediat
Paris paris.orstom.fr	SUN 3/50	Transpac 4 CV	9600 bps	UUCP/F	immediat
Noumea noumea.orstom.fr	SUN 3/260	Transpac par RTC	1200 bps	UUCP/F	2 fois par jour
Seychelles seychel.orstom.fr	TOSHIBA 5200 SOUS 386IX	RTC	1200 bps	UUCP/G	1 ou 2 fois par jour
Bamako bamako.orstom.fr	LEANORD S/ 386IX	RTC	1200 bps	UUCP/G	1 fois par jour
Ouagadougou ouage.orstom.fr	G5 SOUS 386IX	RTC	1200 bps	UUCP/G	1 fois par jour
Dakar (hann) dakar.orstom.fr	SUN 3/60	Senpac 2 CV	2400 bps	UUCP/F	2 fois par jour
Dakar CRODT crodt.orstom.fr	SUN 3/260	Senpac 2 CV	2400 bps	UUCP/F	1 fois par jour
Lome	SUN 4/65C	Togopac ls V22	1200 bps	UUCP/F	1 fois par jour

FIGURE II.1

RESEAU INFORMATIQUE ORSTOM A DAKAR



EXEMPLE DE CONFIGURATION D'UN RESEAU PUBLIC X25

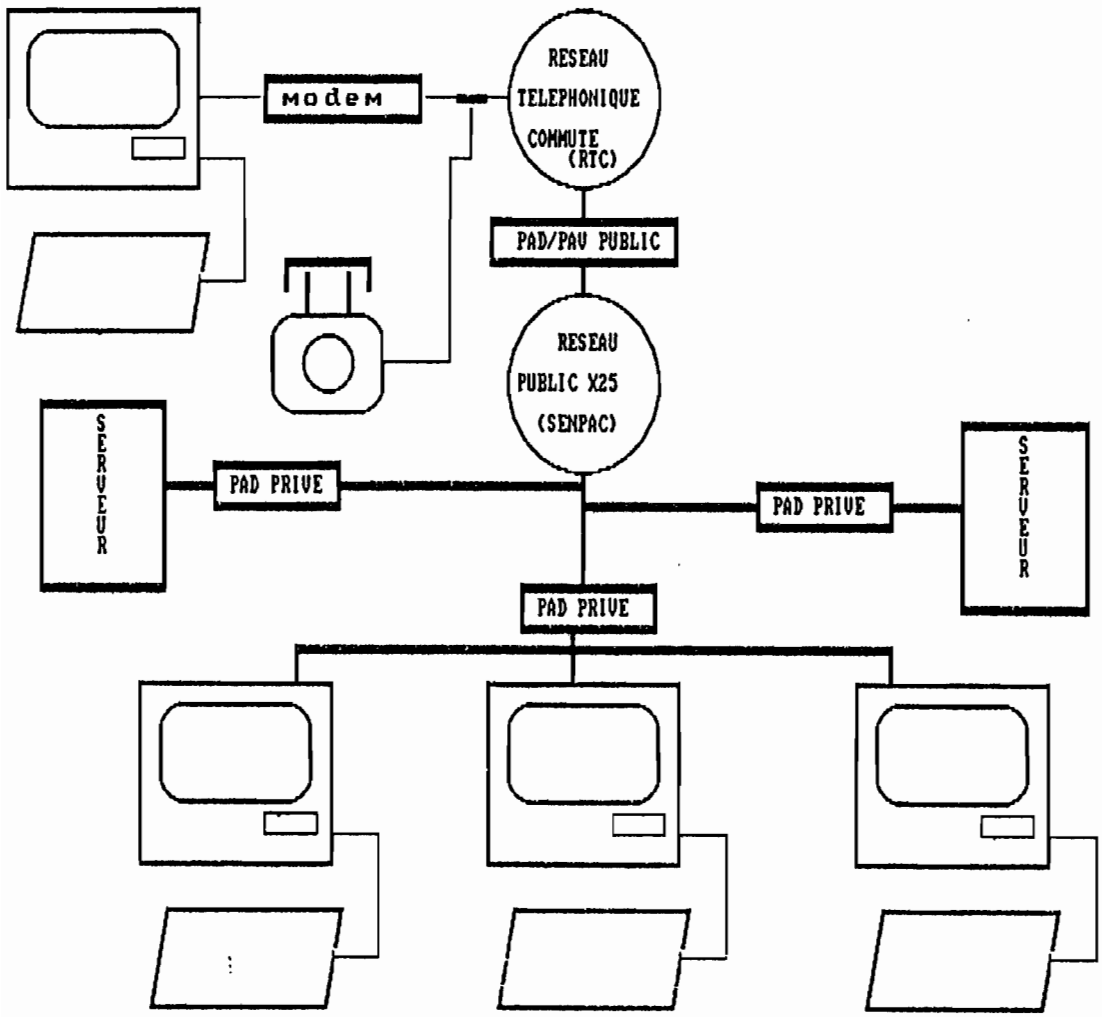


Figure IV 3

A N N E X E 2

LES FENETRES DE DIALOGUES WINDOWS, ET LE SCHEMA DE CONTRUCTION D'UNE APPLICATION WINDOWS.

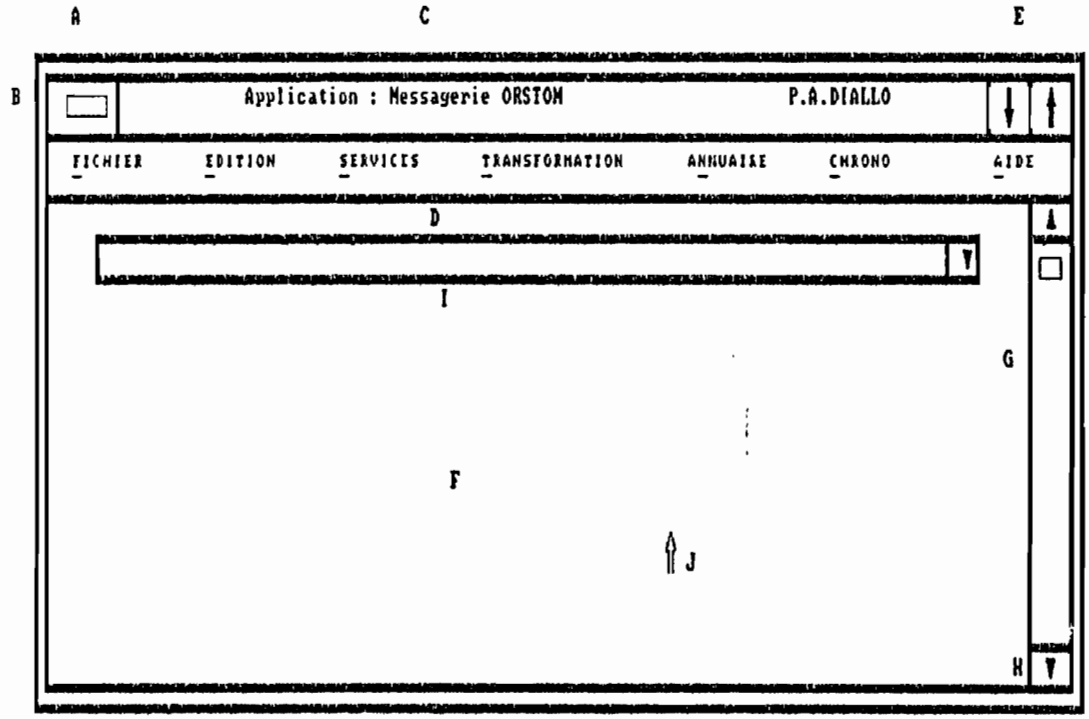


Figure : IX.1

LES DIVERS ELEMENTS DE LA FENETRE MERE DE MESSOR

A : BORDURE DE LA FENETRE MERE

B : CASE DU MENU SYSTEME

C : BARRE DE TITRE

D : BARRE DE MENU

E : CASES REDUCTION AGRANDISSEMENT

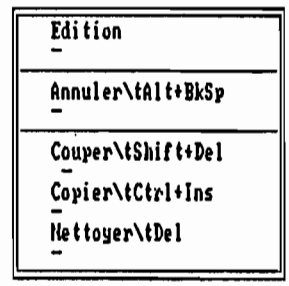
F : ESPACE DE TRAVAIL

G : BARRE DE DEFILEMENT VERTICALE

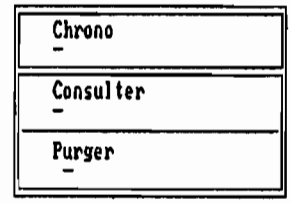
H : CASE DE DEFILEMENT PAR LIGNE

I : FENETRE D'AFFICHAGE DES ENTETES DE MESSAGE

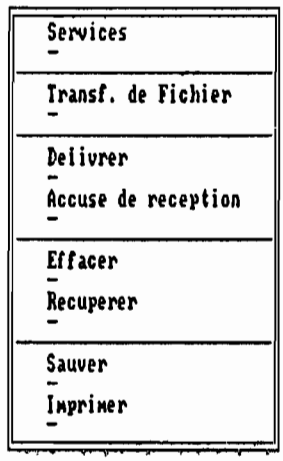
Menu Message



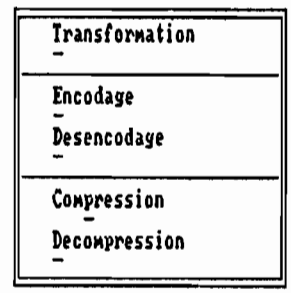
Menu Edition



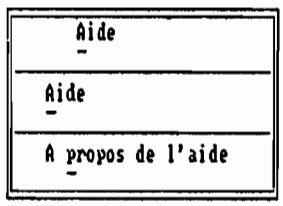
Menu Chrono



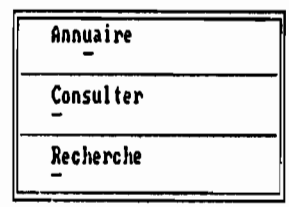
Menu Services



Menu Transformation



Menu Annuaire



Menu Annuaire

Figure : IX.2

LES DIFFERENTES MENUS DE LA FENETRE MERE

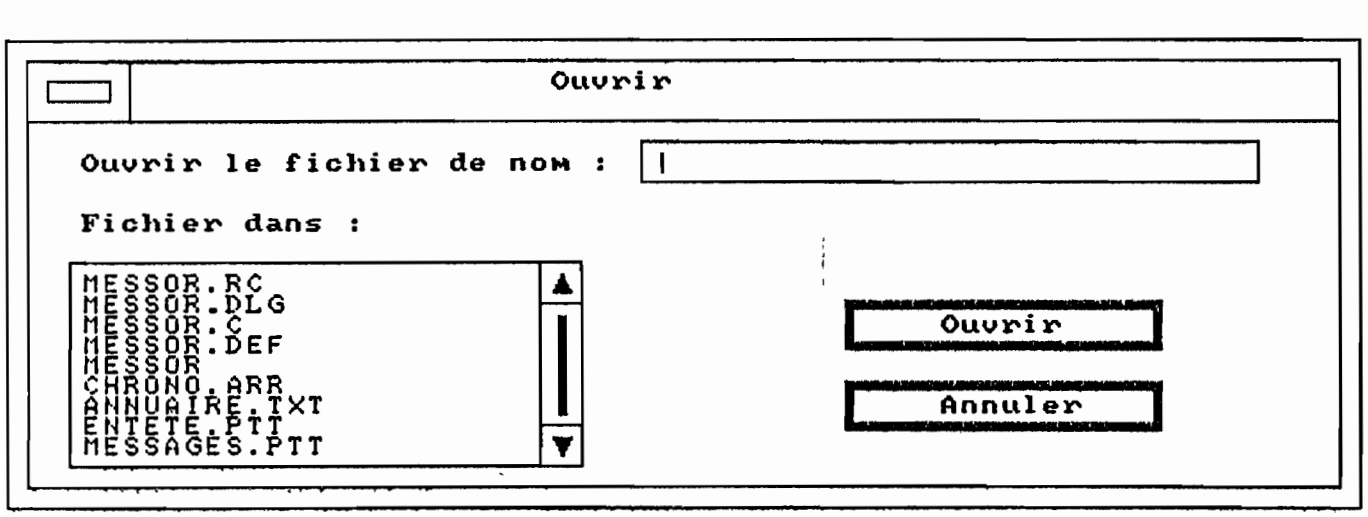


Figure : IX.3 Ouverture d'un fichier sous dos

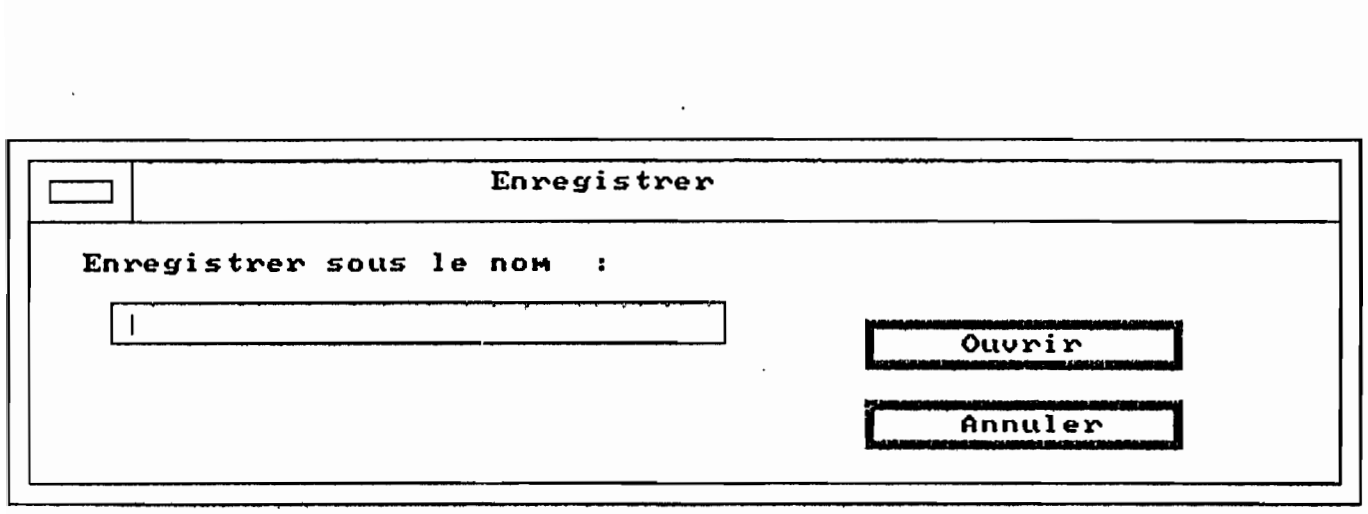


Figure : IX.4 fenetre de saisie de nom de fichier pour la sauvegarde

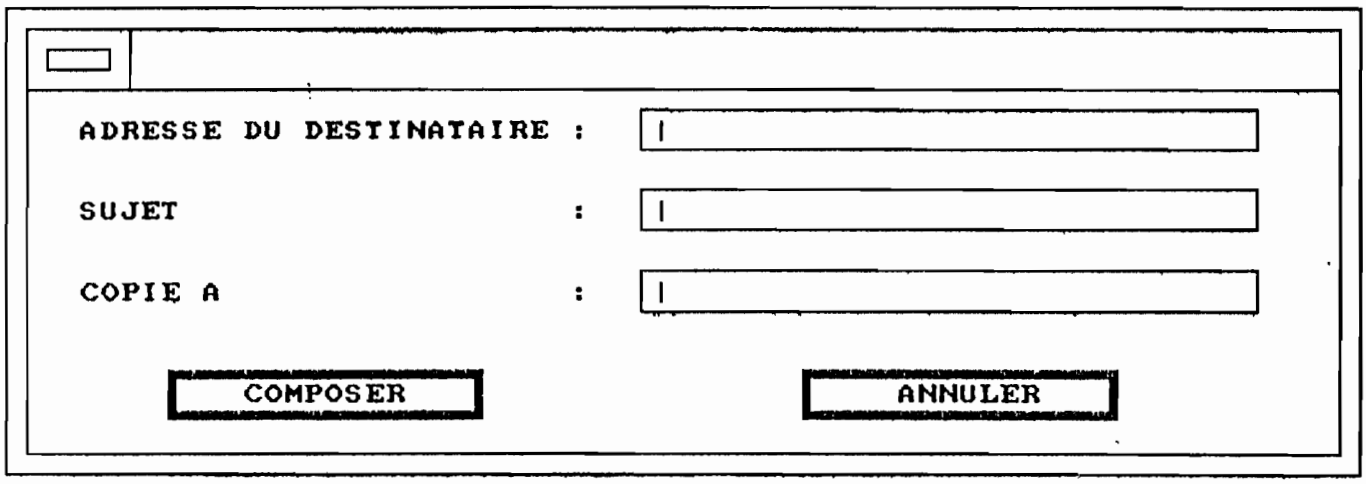


Figure : IX.5

ADRESSAGE DE MESSAGE PERMETTANT D'ACCEDER
A LA FENETRE D'EDITION POUR LA SAISIE.

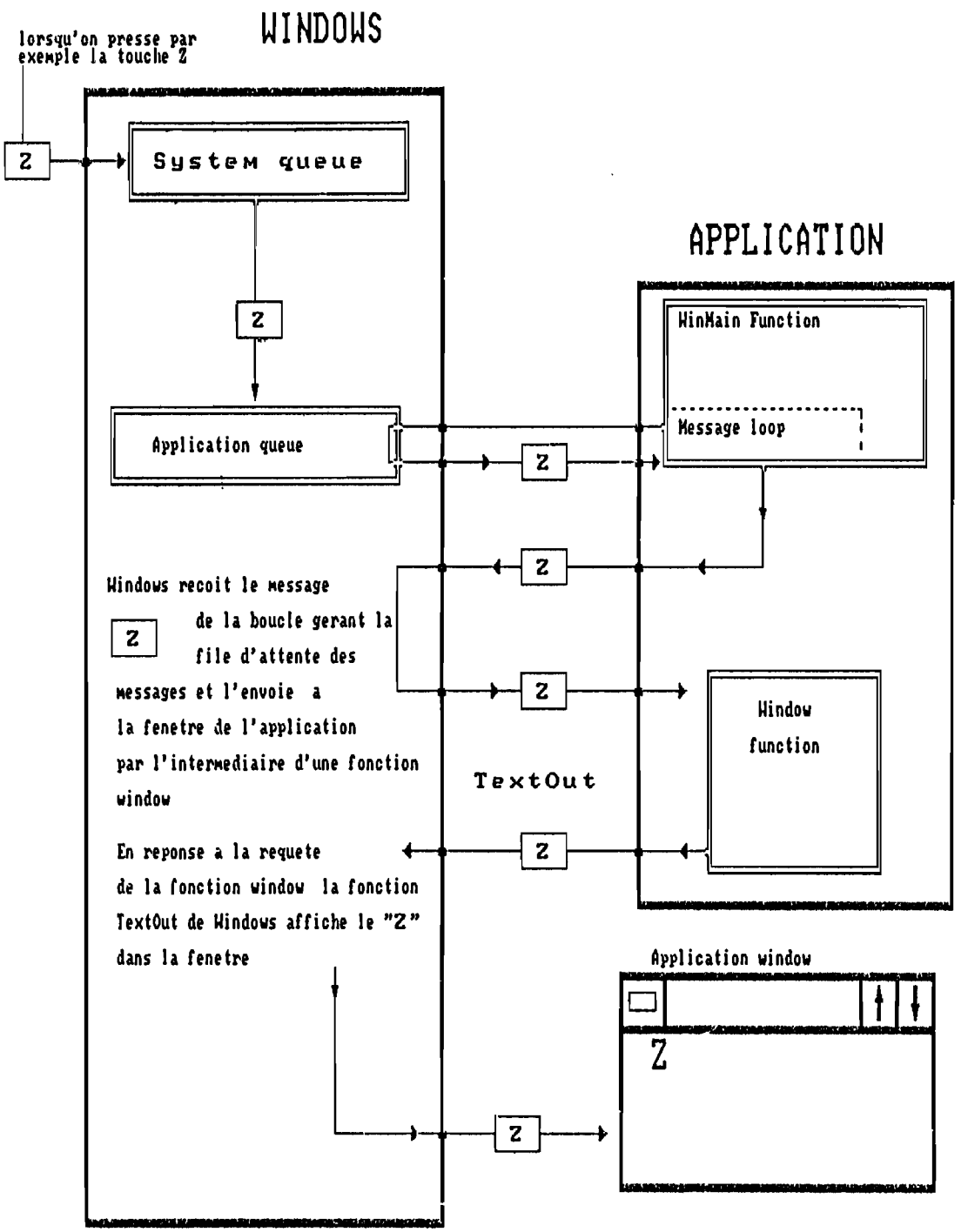


Figure : IX.6

- LES DIFFERENTES PHASES DE TRAITEMENT D'UNE ENTREE AU CLAVIER

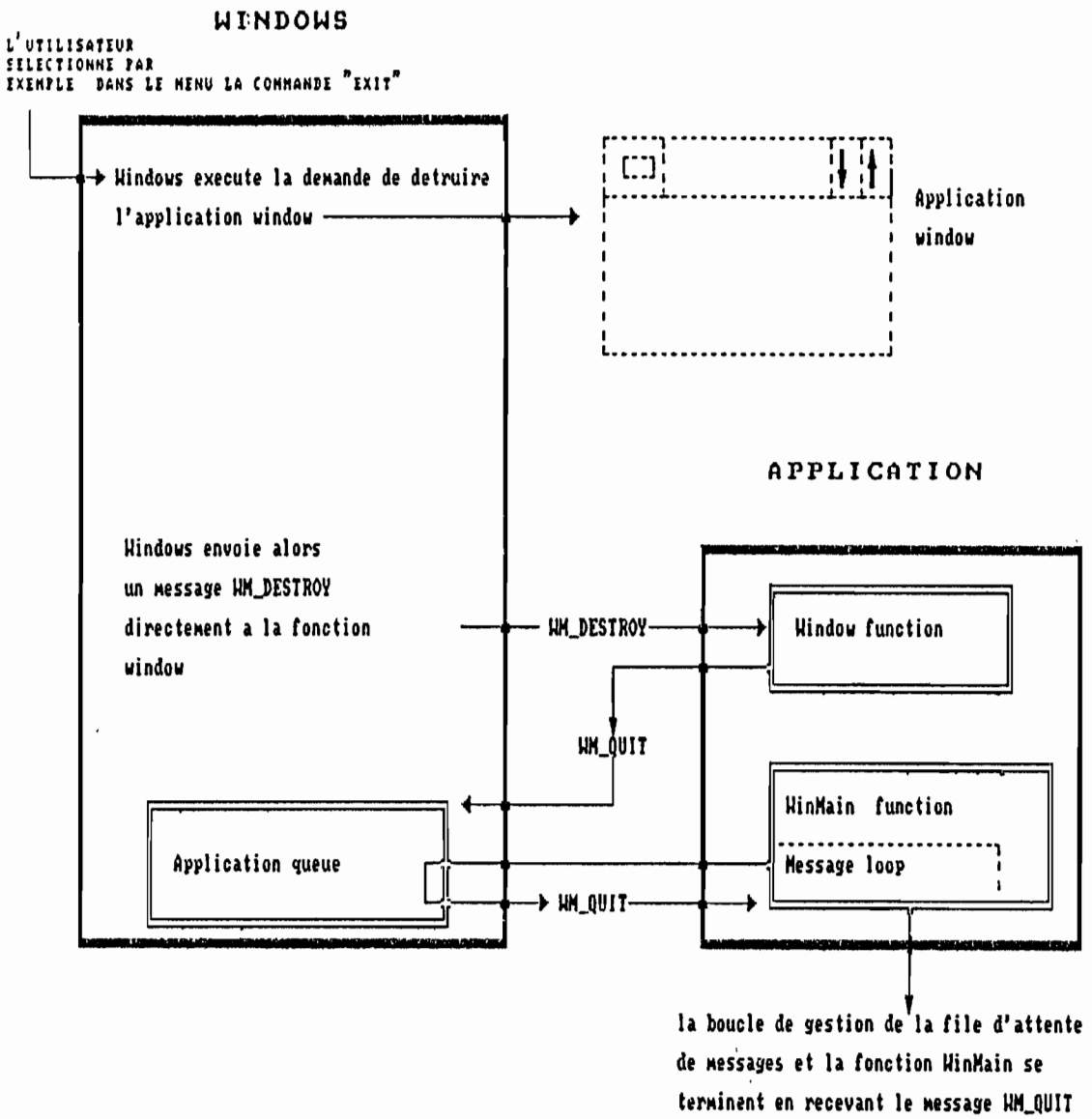


Figure : IX.7

gestion des commandes du menu
quand la boucle de gestion des messages recoit le WM_QUIT
, la boucle se termine et la fonction principale WinMain
procede a la sortie.

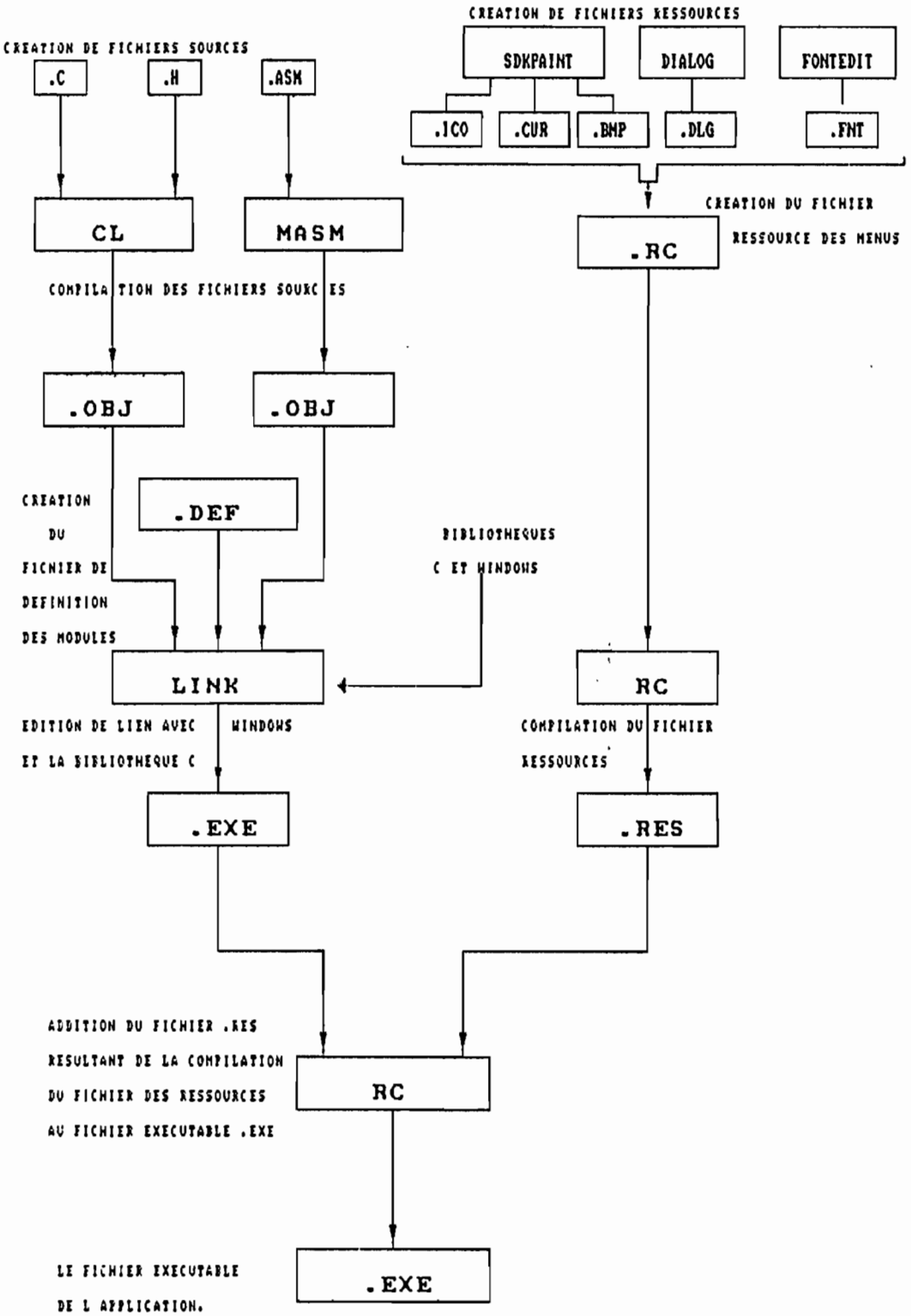


Figure IX.8

Construction d'une application
Windows

A N N E X E 3

"GLOSSAIRE"

GLOSSAIRE

ANALOGIQUE : qualifie le type du signal utilisé. Un signal est dit analogique lorsque la grandeur physique qui le représente subit des variations continues.

APPEL : processus consistant à émettre des signaux d'adresse en vue d'établir une liaison entre stations de données.

ARCHITECTURE (de réseau) : paramètres conduisant à la définition et la structuration d'un réseau de télécommunication. Celle-ci peut être centralisée, étoilée, maillée, répartie, etc...

ARPA : réseau américain de transmission de données fonctionnant selon les techniques de la commutation par paquets.

ASCII : Américain Standard Code for information interchange. Code utilisé pour l'échange d'informations, recommandé par un avis CCITT . Code à 7 bits, plus un bit de parité.

ASYNCHRONE : voir transmission.

AVIS DU CCITT : normes concernant ies télécommunications, émises par le Comité Consultatif international du télégraphe et du téléphone siégeant à Genève. Il existe un nombre important d'avis. Les avis de la série V sont relatifs à la transmission de données sur réseau téléphonique et ceux de la série X à la transmission de données sur les réseaux publics pour données. Parmi les plus connus, citons :

- **l'avis V 24** : il définit les types de signaux devant être échangés sur une jonction entre ETTD et ETCD pour des vitesses de transmission <= 19 200 bit/s.
- **l'avis V 35** : il concerne les vitesse de transmission >= 19.200 bit/s.

- **l'avis X 25** : il précise les normes pour le protocole d'accès aux réseaux publics de commutation par paquets (ensemble de règles d'échange à respecter pour l'utilisation de ces réseaux).

BASE DE DONNEES : ensemble stable et complet d'informations, rapidement accessibles et facilement exploitables, intéressant diverses applications.

BAUD : unité de rapidité de modulation. Si la modulation est effectuée de façon régulière toutes les T secondes (plus petit intervalle entre deux transitions, durée du moment élémentaire) la rapidité de modulation R est donnée par la formule :

$$R = 1/T \text{ (secondes)}$$

BIT : (abréviation de binary digit) - en théorie de l'information, c'est l'unité de la quantité d'information. En pratique, c'est le signal élémentaire utilisé en représentation numérique binaire de l'information. Il peut avoir deux états : 0 et 1.

BIT DE CONTROLE : comprend des bits de parité, de service...

BIT DE PARITE : bit associé à un caractère ou à un bloc de caractères en vue de contrôler l'absence d'erreurs au sein de ce caractère ou de ce bloc après transmission.

BIT DE SERVICE : bit supplémentaire ajouté aux bits d'informations pour permettre l'échange des données.

BIT PAR SECONDE : (Bps) Unité de mesure qui définit le débit d'information (vitesse instantanées d'émission des symboles binaires élémentaires).

BLOC : groupe de bits ou de caractères transmis comme un tout, auquel est généralement appliqué une méthode de contrôle des erreurs.

BUFFERS : (mémoire tampon dans un ordinateur) permet le stockage provisoire et limité des informations. Situé entre l'équipement terminal et la ligne de transmission, il permet une adaptation de l'un à l'autre.

CCITT : Comité consultatif international du télégraphe et du téléphone (voir avis CCITT).

CIRCUIT VIRTUEL : voie de communication réalisée dans un réseau de transmission de données exploité en technique de commutation par paquets. C'est une relation logique établie par le réseau entre stations de données pour assurer le transfert des données. Il remplace ainsi le service assuré traditionnellement au moyen d'une liaison physique.

CODAGE : représentation d'informations par d'autres informations selon un processus de correspondance donné (le code).

COMMUTATION (en transmission de données) : établissement provisoire d'une liaison entre un terminal demandeur et un terminal demandé.

COMMUTATION DE CIRCUITS : processus qui relie à la demande et pendant la durée de la communication deux ou plusieurs terminaux par l'utilisation exclusive d'un circuit de transmission de données.

COMMUTATION DE MESSAGES : technique de transfert d'un message entre deux ou plusieurs terminaux par réception, stockage intermédiaire et retransmission des informations sans pour cela constituer nécessairement un circuit.

COMMUTATION DE PAQUETS : technique particulière de commutation de données dans laquelle l'aiguillage et la circulation des données sont faits par des ordinateurs, après analyse d'informations de service associées à de petits blocs de données constituées en "paquets".

CONCENTRATEUR (de données) : unité regroupant le trafic d'un certain nombre de terminaux sur un nombre plus restreint de voie de communication.

CONNECTE : état d'un organe périphérique relié à une unité centrale, mais aussi nature de l'opération d'entrée ou de sortie de données exécutées sur un organe périphérique relie à l'unité centrale.

CONSOLE (de visualisation) : organe périphérique ou équipement terminal disposant d'un écran de visualisation. Il permet l'affichage en clair des informations échangées avec le système informatique.

CONVERSATIONNEL (mode) : mode d'utilisation d'un système de traitement basé sur le dialogue, à partir d'un terminal, et dans lequel alternent les messages entrés par l'opérateur et les réponses fournies par le centre de traitement.

DECODAGE : passage de l'information reçue codée, à l'information initiale avant codage.

DEMODULATEUR : voir modulateur.

DIALOGUE : voir conversationnel.

EARN : Europeen Academic Rechearch Network

ECRAN DE VISUALISATION : voir console.

EIA : Electric Industries Association

EN TETE : partie initiale
de service.

ETCD : équipement de terminaison de circuit de données. Unité fonctionnelle d'une station de données qui établit une liaison, le maintient et y met fin et qui assure les fonctions nécessaires au transcodage ou à la conversion de signaux entre terminal et ligne de transmission de données.

ETTD : équipement terminal de traitement de données. Unité fonctionnelle d'une station de données pouvant être utilisée comme source de données, assurant le contrôle du transfert d'information.

FNET : réseau français de l'AFUU (Association Française des Utilisateurs d'UNIX). L'AFUU fait partie d'EurOpen, association européenne équivalente et FNET d'EUnet son réseau européen.

FTP (File Transfert Protocole) : transfert de fichier en mode interactif.

FULL DUPLEX : duplex intégral - Voir transmission.

HALF DUPLEX : (semi duplex) Voir transmission.

HANDLE : un entier utilisé par Windows pour identifier un objet créé par une application.

INRIA : Noeud principal du réseau internet français

IP : Internet Protocole.

ISO : International Standard Organisation. Dépend de l'ONU et est chargé de la normalisation dans tous les domaines.

OSI : Open System Interconnection

LIAISON SPECIALISEE (LS): liaison établie sans passer par les installations de commutation, à l'usage exclusif d'un certain nombre de stations de données, donc d'utilisateurs.

LOGICIEL : ensemble des programmes d'ordinateur qui permettent de le faire fonctionner en donnant les instructions nécessaires à la réalisation et à l'enchaînement de tâches élémentaires.

MAINTENANCE : néologisme pour désigner l'entretien des systèmes utilisés. La maintenance est essentiellement préventive. Elle s'effectue par l'emploi de tests méthodiques. Elle se réalise aussi bien sur des équipements informatique que sur les équipements de télécommunication.

MATERIEL : traduction du terme anglo-saxon Hardware. Ensemble des équipements informatiques (ordinateur, mémoires auxiliaires, imprimante...) d'un centre de traitement de l'information.

MEMOIRE : Partie du système informatique où l'information peut être stockée et atteinte en permanence. On distingue la mémoire centrale et les mémoires auxiliaires distinctes de l'unité centrale. (Bandes magnétiques, disques et tambours magnétiques.

MESSAGE : ensemble organisé d'informations constituant un tout. Celui-ci comprend un groupe de caractères et une suite d'éléments binaires de commande. Est véhiculé de la source de données vers le collecteur de données.

"MESSAGE" : les guillmets permettent simplement de faire la différence entre un message Windows et les autres types de messages.

MODEM : modulateur - démodulateur - organe fonctionnel servant à moduler les signaux. La fonction essentielle du modem est de permettre la transmission de données numériques sur des circuits analogiques.

MODULATEUR : appareil effectuant la modulation d'un signal à transmettre.

MODULATION : opération par laquelle une ou plusieurs caractéristiques d'une onde (fréquence porteuse) sont modifiées, par le signal (analogique ou numérique) à transmettre. Suivant que l'on fait varier l'amplitude, la fréquence ou la phase de la fréquence porteuse, on obtient de la modulation d'amplitude, de fréquence ou de phase.

MULTIPLEXAGE : technique de transmission qui permet le transport simultané de plusieurs signaux sur une même ligne.

MULTIPLEXEUR : organe effectuant le multiplexage soit en fréquence, soit de façon temporelle, permettant à plusieurs sources de données d'utiliser simultanément des moyens communs de transmission tout en assurant à chaque source sa propre voie indépendante.

NOEUD : point où, dans un réseau de transmission de données, une ou plusieurs unités fonctionnelles, sont mises en relation avec des lignes de transmissions de données

OCTET : groupe de huit éléments binaires utilisés comme une entité.

ORSTOM : Office de Recherche Scientifique et Technique d' Outre Mer. Bien que gardant cette appellation, ce sigle signifie maintenant Institut Scientifique et Technique pour la Recherche en Coopération.

PAD : packet assembler - désassembler. Dispositif d'assemblage et de désassemblage de paquets dont les fonctions essentielles consistent à assembler les caractères de données en paquets de données et à désassembler les paquets en caractères de données. Dans TRANSPAC le PAD assure l'adaptation des terminaux asynchrones mode caractère au réseau.

PAQUET : groupe d'éléments binaires représentant des données et des signaux de commande organisé selon un format déterminé et transféré comme un tout, conformément aux règles d'une procédure de transmission.

REPONSE : processus consistant à donner suite à l'appel d'une station pour réaliser une liaison entre stations de données.

RESEAU (de données) : ensemble des unités fonctionnelles et des circuits de données, entre les terminaux.

RESEAU PUBLIC DE DONNEES : réseau établi et exploité par une administration afin de mettre des services de transmission de données à la disposition du public.

RESSOURCE : en informatique, toute entité pouvant être utilisée par plusieurs utilisateurs différents : mémoires, etc...

R.I.O : Réseau Informatique ORSTOM

ROUTAGE : détermination du chemin emprunté dans le réseau, par une communication ou par un paquet de données. (Ce vocable affecte aussi bien le parcours emprunté par une liaison dans le réseau, que les parcours utilisés par les différents constituants du réseau : CP-GS, etc...).

RTC : Réseau Téléphonique Commuté.

SAISIE DES DONNEES : entrée des données sur système informatique. Elle peut se faire par lots (batch), dans ce cas les données sont regroupées sur un support physique (cartes, rubans magnétiques, etc...) et la transmission se fait en différé ("off lines"").
Elle peut se faire par envois non groupés, on utilise alors une transmission directe ("on line") pour une telle entrée.

SENPAC : Réseau de transmission de données par paquets du Sénégal. Idem que TRANSPAC en France.

SIGNAL : moyen de communication dynamique de l'informatique dont la grandeur, fonction du temps, caractérise un phénomène physique (signe analogique) ou représente des données (signal numérique).

SIGNAL D'ARRET ("Stop") : en transmission asynchrone, signal transmis immédiatement après un caractère pour indiquer la fin.

SIGNAL DE DEPART ("Start") : en transmission asynchrone, signal indiquant le début d'un caractère.

SMTP : Simple Mail Transfert Protocole.

SNA (Systems Network Architecture) : fut introduit par IBM en 1974, il définit les fonctions à exécuter dans un réseau leurs interactions mutuelles.

SOFTWARE : voir logiciel

START : voir signal de départ.

STOP : voir signal d'arrêt.

SUPPORT (de télécommunications) : élément du réseau chargé d'acheminer les signaux contenant l'information.

TCP : Transfert Command Protocole.

TELNET (TErminal NETwork) : accès en mode terminal sur une autre unité centrale, à travers le réseau.

TEMPS D'ACCES : temps s'écoulant dans un système informatique ou téléinformatique entre le déclenchement d'une opération et le résultat de son premier traitement.

TRAME : enveloppe de transmission de l'information. En multiplexage temporel, la trame représente l'ensemble des intervalles de temps, consécutifs, alloués aux différentes sous-voies.

TRAITEMENT (de l'information) : exécution d'opération programmées sur un ensemble d'informations pour extraire d'autres informations. L'ordinateur est actuellement le moyen le plus performant de traitement de l'information. Ce traitement peut se faire en conversationnel ou par lots.

TRANSMISSION (en télécommunication) : ensemble de moyens spécifiques destinés à acheminer les informations (faisceaux hertziens, câbles coaxiaux, organes de multiplexage, etc...)

TRANSMISSION (mode):
- asynchrone : mode de synchronisation de l'émetteur et du récepteur par l'utilisation pour l'émission de chaque caractère, d'un bit "start" le précédant et d'un bit "stop" le terminant.
- de données : acheminement de données d'un point à un autre par l'intermédiaire de signaux transportés par une voie de télécommunication.
- synchrone : mode de transmission des données dans lequel l'instant de l'émission de chaque signal représentatif d'un élément binaire et synchronisé sur une base de temps données.

TRANSPAC (voir SENPAC)

UUCP (UNIX to UNIX Communication Protocole) : ensemble de programmes assurant la transmission de données en système UNIX.

VOIE DE TRANSMISSION : ensemble des moyens permettant la transmission de signaux.

REFERENCES BIBLIOGRAPHIQUES

Bart Anderson, Bryan Costales and Harry Henderson

UNIX Communications (Howard W.SAMS et company 87).

Charles Berthet Informatique de gestion et Cobol. (Bordas.informatique 87)

Deviller Y Mise en service d'une messagerie UNIX (AFFU 88).

Devillers Y Le réseau Fnet-EUnet Documentation technique Fnet - Nov 90.

Dirlewanger,R Installation de Sunlink en vue d'une connexion IP/X25 vers INRIA (INRIA 90)

Donnalyn Frey, Rich Adams

A Directory of Electronic Mail (Adressing and Networks 90)

Dougherty Dale, Tim O'Reilly

DOS meets UNIX, A departemental Computing Perspective (Nutshell Handbook 87)

Douglas E. Cormer Internet working with TCP/IP - Principes, Protocoles and Architecture

Dumas Dominique,Pinse Dominique

Le réseau EARN (90)

Fontaine A.B, Ph. Hammes

UNIX Système et environnement (Masson 86)

Galacsi C. Système d'information et Bases de données (Bordas Informatique 87)

Kermighan Brian, Denis M. Ritchie

Le Langage C, (2° édition 90)

Meyer JJ. Pratique du Turbo Pascal (Edition Radio)

O'Reilly Tim, Todino Grace

Managing UUCP and USENET (Nutshell Handbook 86).

O'Reilly Tim Using UUCP and USENET (Nutshell Handbook 86).

Pascal Renaud, Monique Michaux

Le RIO : Un réseau international de la recherche dans les pays en voie de développement 91

Petzold Charles Programming Windows 3, 2nd édition (Microsoft press 90)

Pujolle G., Horlait E.

Architecture des réseaux informatiques (Eyrolles 90).

Pujolle G., Seret D., Dromard J., Horlait E.

Réseaux et Télématique, Tome 1 (Eyrolles 87)

Pujolle G., Seret D., Dromard J., Horlait E

Réseaux et Télématique, Tome 2

R and D E-Mail directory (EUUG).

Ran Couran R. C avancé (Collection Marabout 90)

Rebecca Thomas A User Guide to the UNIX System (McGrow Hill)

SMTP Simple Mail Transfert Protocol.

ARPA Internet Text Messages

Standard for the format.

Documentation technique CEDIA-Passerelle

Le réseau Fnet-EUnet (INRIA 90).

Microsoft Windows Software Development Kit

Reference - volume 1

Reference - volume 2

Guide to Pprogramming

Tools (Microsoft 90)

Sun Microsystems Sunlink TM X25 System Administration Guide (June 87)