Ce cours va donc comprendre une courte introduction, quelques techniques de bases et des exemples dans le désordre en attendant que je fasse la synthèse (pour l'année prochaine?)

Acheter livres DCE, Acteurs?, NTP : le temps de trasnmission, panne, mise a l'heure correction, NNTP, RIP, PVM, Transactionnel,

DCE : 1 objet = 1 serveur, on exporte des interfaces, on fournie des moyens de se connecter aux serveurs. Le services minimal est celui qui permet d'intéroger sur les services existants.

Open Database Connectivity (ODBC) is a set of database drivers that enables connectivity among different databases. In client/server environment, ODBC enables your client application to connect to virtually any database engine, thus allowing you to work with heterogenous database engines without changing much of your front-end application.

OLE: Objet communiquant integrable COM: Component Object Model L'objet peut être une agrégation d'objets. Un objet peut-être utilisé par plusieurs autres applications en même temps. Les objets peuvent être dans le même processus ou bien séparé même sur des machines différentes (RPC). Interface : Nom et ensemble de fonction applicable (fenetre graphique par exemple). OLE donne des fonctions de base qui peuvent être redéfinies. On peut évidemment définir des objets non-graphiques/interactif (transfert de données, calcul, ...)

CORBA: Common Object Request Broker Architecture Soutenu par l'OMG.

PVM: Moyen communication entre objets.

A AJOUTER : Peer to Peer (GNUTella) : tout serveur

A AJOUTER : Serveur d'application.

# Table des matières

# 1 Introduction : Architectures client/serveur

Afin de vous faire découvrir l'architecture client/serveur de manière simple, on va étudier un cas particulier et voir les apport de l'approche CS dans ce cas.

Considérons une application permettant de visualiser des documents (manuels, plans, annuaires, dictionnaires, ...)

## 1.1 Architecture monolithique

L'application est composée d'un seul programme. Celui-ci lit les informations localement (disque dur, CD ROM, ...) et gère l'interaction avec l'utilisateur.

Problèmes :
- Les données ne sont pas partageable, elles sont dupliquées.
- Le logiciel est «fermé», difficilement modifiable.

## 1.2 Client/Serveur

L'architecture client/serveur est la suite logique de la programmation structurée. En effet après avoir séparé les fonctionnalités à l'intérieur d'un programme, on peut *découper l'application en plusieurs programmes distincts offrant chacun un service différent.*

Les communications à l'intérieur d'un programme se font par appel de fonctions, ce qui est important est l'API. *Les communications entre services d'un programme respectent un protocole.* C'est une extension de l'abstraction procédurale.

Le communications entre services peuvent utiliser de nombreux mécanismes différents et notamment passer par le réseau, elles ont aussi de nombreux attributs :
- synchrone.
- persistante.
- fiable.
- securisée.

Nous allonc voir les plus grand avantages de l'architecture client/serveur dans l'exemple nous concernant.

- Le client est un programme s'occupant de l'interface utilisateur.
- Le serveur s'occupe de lire les informations pour les envoyer à qui le demande.
- Les deux communiquent grâce à un protocole.

Avantages :
- *Partage des données* ; en effet le client et le serveur peuvent être sur des machines différentes.
  - Plusieurs client peuvent accéder au même serveur.
  - Un client peut accéder à plusieurs serveurs simultanément.
- *Ouverture* ; le protocole est souvent simple et permet de faire évoluer les programmes *sans avoir besoin de disposer des sources du client ou du serveur* :
  - En réécrivant le client ou le serveur.
  - En ajoutant de nouveaux programmes respectant le protocole.
  - Si l'on a $n$ programmes client et $m$ programmes serveurs cela revient à avoir $nm$ applications différentes!
- *Fiabilité* ;
  - La complexité d'un programme est grossièrement en $O(n^2)$ et découpant le programme en deux on divise la complexité totale par 2! Le programme est plus propre et donc plus facilement modifiable. (De la même manière que l'on ne fait jamais de fonctions faisant des entrées/sorties et des calculs, ou utiliser des variables globales.)
  - A nombre de bug constant par ligne, un programme deux fois plus long contient donc deux fois plus de bug ; mais la probabilité qu'un bug soit néfaste elle n'est pas linéaire. En coupant le programme en deux, les bugs de l'un des programmes ne risquent pas de déclencher des bugs dans l'autre programme.
  - En cas d'arrêt du programme serveur, le client peut s'adresser à un autre serveur sans que l'utilisateur ai à relancer son application. Le plantage d'une des parties du programme ne fait pas planter les autres. Il faut par contre synchroniser les serveurs afin qu'ils diffusent les même données.
  - *Facilité de programmation* ;
    - On peut facilement débugger les programme en espionnant leurs échanges de données.
    - On peut tester indépendamment les deux parties et même les faire réaliser par des équipes différentes.
- *Minimisation* ; Le programme client étant plus petit il nécessite des machines moins puissantes pour fonctionner.

- *Mise à jour des logiciels et évolutions*;
  - On peut faire évoluer le programme serveur sans avoir besoin de mettre à jour tous les programmes clients et inversement.
  - On peut ne diffuser le programme serveur que pour certaines architectures de machines, pas forcément toutes (contrairement au client)
  - On peut offrir des interfaces utilisateurs variées sans avoir besoin de modifier le protocole.
  - On peut faire évoluer le serveur afin qu'il puisse lire de nouveaux type de documents (ou de nouveaux support d'information) sans avoir besoin de faire évoluer le client.
- *Parrallélisme*;
  - Le client et le serveur peuvent fonctionner sur des processeurs différents.
  - Un client peut utiliser plusieurs serveurs différents.

Inconvénients :
- Il faut définir un protocole ; certain outils simplifie la tâche.
- L'utilisation d'un protocole ralenti l'application.
- L'utilisation d'un réseau introduit des temps de latence.
- Le serveur peut être surchargé de CPU.
- Le serveur peut être surchargé de disque.
- Le serveur peut être surchargé de réseau.
- Les risques d'espionnage.

Comment diminuer les surcharges ?

## 1.3 Cache

On garde l'information centralisée. Les clients s'adressent à des serveurs caches qui eux-même s'adressent à d'autre serveurs caches ou au serveur centrale.

Les caches stockent les informations qu'ils font passer. Si ont leur demande une information qu'ils possèdent dans leur cache, ils ne demandent pas à leur supérieur.

Les informations cachées sont invalidées :
- Si elle sont trop vielles et qu'il n'y a plus de place dans le cache.
- Si après vérification auprès du serveur elles ne sont plus à jours.
- Si le serveur nous préviens qu'elles ne sont plus à jours. (solution complexe)

Cela ne nécessite pas d'ajouter un nouveau protocole.

Avantages :
- Soulage le serveur de boulot
- Diminue le traffic réseau vers le serveur.
- Accès beaucoup plus rapide (car proche) si l'information est dans le cache.
- On peut diminuer la taille du serveur (c'est important car le prix d'un machine est en $O(2^n)$)

Inconvénients :
- Il faut de la place disque pour stocker le cache
- La latence est plus grande si rien n'est dans le cache.

L'accès à un document peut passer par un cache de ville puis de pays avant d'arrive au serveur central.

## 1.4 Répartition

Les données sont réparties sur plusieurs serveurs. Le client à partir de la clef de la donnée en déduis à quelle machine il doit s'adresser

Avantages :
- Moins de disque sur chaque serveur
- Répartition de la charge du serveur en plusieurs serveurs (diminution taille serveur)
- On peut faire du parralélisme en récupérant des données de plusieur endroits en même temps.

Inconvénients :
- Non diminution de la bande passante totale.

## 1.5 Cache bidirectionnel

Les données sont réparties. Si l'on utilise un cache standard, cela diminue la charge réseau coté demandeur mais non du coté serveur qui reçoit des demandes du mondes entier quand elle ne sont pas dans un cache distant.

Pour résoudre le problème :
- On défini (ou il s'auto défini) le graphe des caches et des serveurs.

- On fait disparaitre la notion de cache de niveau supérieur.
- L'information circule de cache en cache par le chemin le plus court.

Par exemple : Mon information répartie est à NY, elle passe par le cache de Lyon, le cache de la France, le cache Américain, le cache de NY.

Si l'information est souvent demandée à l'extérieur des états unis alors l'information restera dans le cache Américain et l'on n'encombrera pas le réseau de NY pour demander cette info.

Les avantages sont ceux de la répartition et du cache et les inconvénients la latence et la place prise par le cache.

### 1.6 Latence et Asynchronisme

Pour diminuer les temps de latence, il faut utiliser des protocoles asynchrone ou le client et le serveur s'échangent des informations sans attendre une réponse immédiate de l'autre.

C'est un protocole événementiel qui n'est pas forcément plus complexe à programmer mais qui multipliera les performances.

### 1.7 Architecture 3/3

C'est une architecture ou le client ne demande pas au serveur mais à un intermédiaire.

### 1.8 Architecture tous/sans serveur

Architecture ou chaque client est associé à un serveur sur la même machine et ou les ressources sont réparties automatiquement/dynamiquement entre les clients/serveurs de manière transparente.

C'est le futur des systèmes de fichiers.

### 1.9 Les problèmes du clients/serveur

Pour le protocole :

- Définition : Pétri, Loto
- Outils : TCPIP, Socket, RPC, DCE
- Sécurité : Secure Socket, Kerberos, DCE
- Fiabilité : Reprise en cas d'arrêt ou protocole sans état.
- Vitesse : asynchronisme, compression, prédictions
- Conseil :
  Dans les communications, il faut être libéral dans ce que l'on reçoit et conservateur dans ce que l'on envoit.
  Le protocole doit être évolutif, on doit pouvoir lui ajouter des fonctionnalités dans le futur.

Pour l'architecture :

- Définition : Simulation
- Outils : Cellule DCE
- Sécurité : Capacités.
- Fiabilité : Redondance
- Vitesse : Cache
- Conseil :
  Evolutivité.

# 2 Introduction : pourquoi le client/serveur?

L'architecture client/serveur est la suite logique de la programmation structurée. En effet après avoir séparé les fonctionnalités à l'intérieur d'un programme, on peut *découper l'application en plusieurs programmes distincts offrant chacun un service différent.*

Les communications à l'intérieur d'un programme se font par appel de fonctions, ce qui est important est l'API. *Les communications entre services d'un programme respectent un protocole.* C'est une extension de l'abstraction procédurale.

Le communications entre services peuvent utiliser de nombreux mécanismes différents et notamment passer par le réseau, elles ont aussi de nombreux attributs :

- synchrone.
- persistante.
- fiable.
- securisée.

Nous allonc voir les plus grand avantages de l'architecture client/serveur.

## 2.1 Programmation

Le fait de découper un programme en modules offrant des services différent apporte plusieurs choses importantes au programmeur :

- Le programme est plus propre et donc plus facilement modifiable. (De la même manière que l'on ne fait jamais de fonctions faisant des entrées/sorties et des calculs, ou utliser des variables globales.)
- Grossièrement, la probabilité de plantage d'un programme est en $O(n^2)$ en effet plus un programme est gros plus il y a de bug, mais de plus ceux-ci ont de plus en plus de chance de devenir actif. Ceci n'est vrai que si les processus s'exécutent avec des espaces d'adressages séparés.
- On peut facilement débugger les services un par un et espionner leurs échanges de données.

On peut dire que les commandes UNIX sont des services et le pipe le moyen de communication (monodirectionnel) permettant l'échange de données entre services.

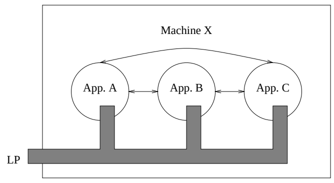
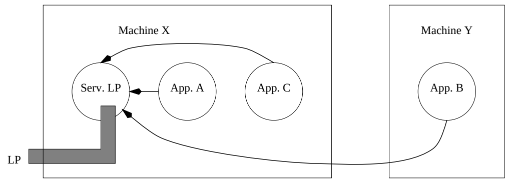
Evidemment, les performances sont moindres mais les possibilités de combinaisons sont infinies.

## 2.2 Gestion d'une ressource

C'est le cas le plus fréquent, il apparaît dans de nombreux domaines :

- Périphériques : Disque, Ecran, Liaison, CPU
- Services : Les mêmes à un niveau plus élevé de sémantique.

En résumé :

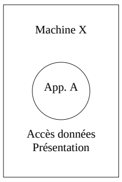
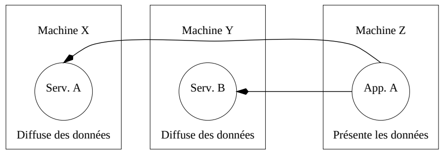
| Monolithique | Client/Serveur |
|---|---|
|   |   |
| Inconvénients :<br>– Programme complexe pour partager la ressource.<br>– Si une application bugge, cela perturbe les autres.<br>– Le conducteur (driver) imprimante est dupliqué. | Avantages :<br>– Pas les inconvénients du monolithique.<br>– On ne peut pas détourner la resource.<br>– Asynchronisme possible.<br>– Imprimante distante possible.<br>– Traitement répartis :<br>moins de travail pour le serveur (donc plus petit)<br>présentation plus agréable sur le client (interactivité, graphisme, ...)<br>– Evolutif : changement de type d'imprimante, imprimantes en parallèle, imprimante virtuelle (PostScript).<br>– Testable indépendamment.<br>Inconvénients :<br>– Il faut définir un protocole (`rlp`, `smb`)<br>– Plus lent car il faut écrire/lire/vérifier le protocole.<br>– Si le serveur est monotache, il peut être bloquant. |

Autres exemples :
- Système d'exploitation : toute les ressources physiques.
  On peut considérer que tout système d'exploitation est un serveur et les processus, des clients de ce serveur. Le micro-noyau (`Mach`) est en fait réduit à une gestion de serveurs, le système basé sur `Mach` devient alors une collection de serveurs.
- X11 : ressource écran (niveau supérieur)
- ODBC : ressource base de donnée
- HTTP : ressource WWW
- FTP : ressource FTP
- TELNET : ressource "commande" distante
- ...

## 2.3 Système ouvert

Une fois un protocole de communication client/serveur connu. Alors les programmes clients et serveurs peuvent être développés indépendamment par plusieurs sociétés pour plusieurs matériels pour différents utilisateurs.

L'exemple suivant peut par exemple être une application permettant de naviguer dans une documentation.

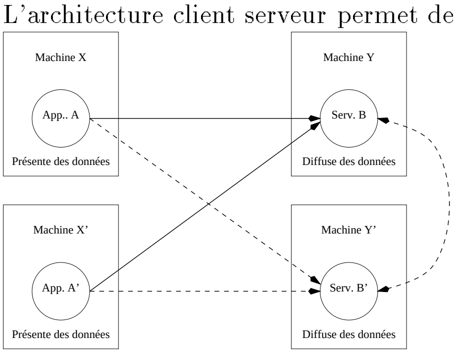
<table>
<tr><th>Monolithique</th><th>Client/Serveur</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>Inconvénients dus au fait que tout est dans le même programme :<br>– L'accès au données dépend du système d'exploitation.<br>– On ne peut pas changer de navigateur.<br>– On a une application par constructeur.</td><td>Avantages :<br>– Nombre d'utilisation différentes : $n * m$<br>Clients : Mosaic, Netscape, Lynx, Arena, ...<br>Serveurs : apache, cernhttpd, ncsahttpd, netscapehttpd, ...<br>– Joindre des ressources n'existant pas sur la même machine, dans ce cas : stockage et affichage.<br>– Le protocol étant standard, n'importe qui peut ajouter des clients et serveurs.<br>– Un client peut interroger différents serveur et présenter un uniformément les résultats.<br>– On peut faire des faux serveurs ou clients, par exemple les proxies.</td></tr>
</table>

En fait l'ouverture est donnée par un protocole standard.

## 2.4 Fiabilité

Dans le cas de panne de machine contenant une application monolithique on ne peut plus rien faire à par changer de machine.

L'architecture client serveur permet de détourner ce problème.

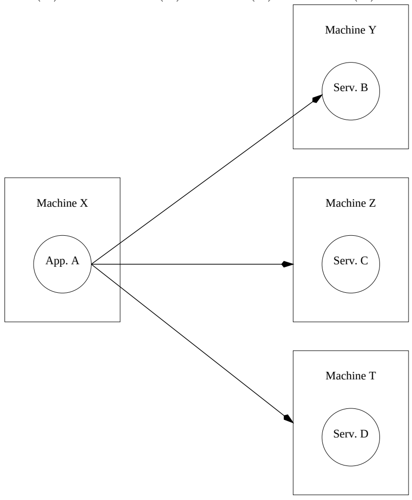



En fonctionnement normal tous les clients utilisent le même serveur. En cas de panne du serveur les clients basculent sur le serveur de secour. Il faut évidemment faire en sorte que les serveurs dialoguent ensemble pour se mettre à jour.

Si un client tombe en panne, rien n'empêche d'utiliser un autre client.

## 2.5 Parallélisme/Down-sizing

Le(s) serveur(s) et le(s) client(s) peuvent être sur des machines différentes.

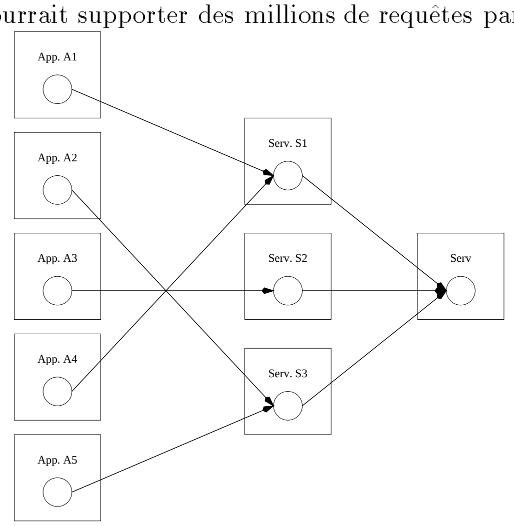



Possibilités :
- Calcul répartie : accès concurrent à des moteurs de recherche WWW par exemple.
- Données réparties : gestion des noms logiques des machines (*names services*)

Donc :
- Machines plus petites (prix en grossièrement $O(2^n)$).
- Plus de puissance.
- Facilité d'extension.

## 2.6 Distributions (3/3)

Ceci est l'architecture classique «two-tiers» mais il existe une architecture «three-tiers» dans le cas de très nombreux clients. Des serveurs intermédiaires répartissent le travail sur d'autres serveurs. En effet quelle machine pourrait supporter des millions de requêtes par secondes?



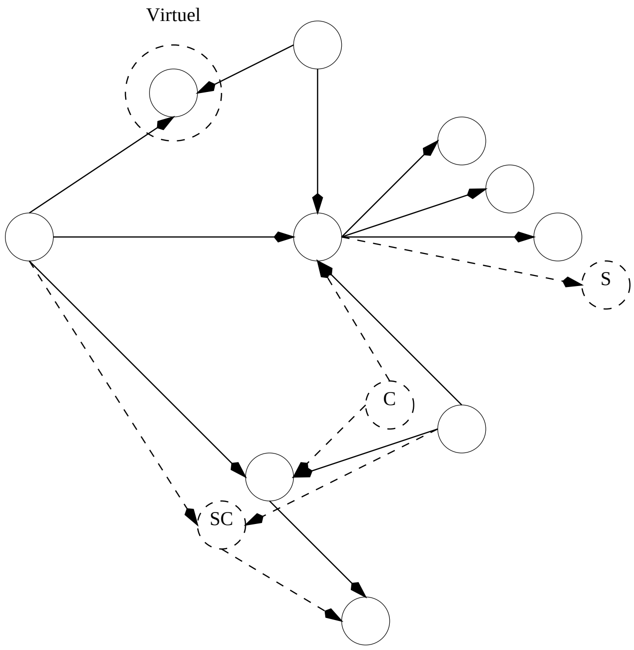

Avantages :
- C'est une méthode pour transformer un serveur synchrone en un serveur asynchrone. En effet le bloquage d'un des serveurs n'embête qu'une partie des clients.
- Un paralèlisme est possible.

Un exemple simple de ce type d'architecture est NFS. Bien que le serveur soit sur une seule machine les serveurs NFS sont plusieurs à accéder au disque de la machine. Cette technique est gagnante même pour une machine mono-processeur.

Cette architecture permet de simplifier les serveurs car ceux-ci sont purement synchrone.

## 2.7 Evolutif

Avant pour évoluer, il fallait changer de machines....




Maintenant :

- Plus de disque : on ajoute un serveur
- Plus de point d'entrée (et donc de CPU) : on ajoute un client
- Décharger le réseau : client/serveur intermédiaire (3/3)

Une architecture optimale ne fait plus la différence entre client et serveurs et l'ajout d'une machine profite à tous quelque soit son utilisation.

## 2.8 Les problèmes du clients/serveur

Pour le protocole :

- Définition : Pétri, Loto
- Outils : TCPIP, Socket, RPC, DCE
- Sécurité : Secure Socket, Kerberos, DCE
- Fiabilité : Reprise en cas d'arrêt ou protocole sans état.
- Vitesse : asynchronisme, compression, prédictions
- Conseil :
  Dans les communications, il faut être libéral dans ce que l'on reçoit et conservateur dans ce que l'on envoit.
  Le protocole doit être évolutif, on doit pouvoir lui ajouter des fonctionnalités dans le futur.

Pour l'architecture :

- Définition : Simulation
- Outils : Cellule DCE
- Sécurité : Capacités.
- Fiabilité : Redondance
- Vitesse : Cache
- Conseil :
  Evolutivité.

# 3 Outils

## 3.1 Graphes de Pétri

Je ne suis pas spécialiste.... Je vous présente une version simplifiée tel qu'elle est utile en informatique.

- noeud : contient des jetons ou non.
- transition : contient une condition et des actions
- Arêtes orientée : entre un noeud et une transition
- Un marquage initial.

Si la condition est vérifiée, un jeton est enlevé de chaque noeud de départ (si ce n'est pas possible alors rien ne se passe) et un jeton est ajouté dans chaque noeud d'arrivé.

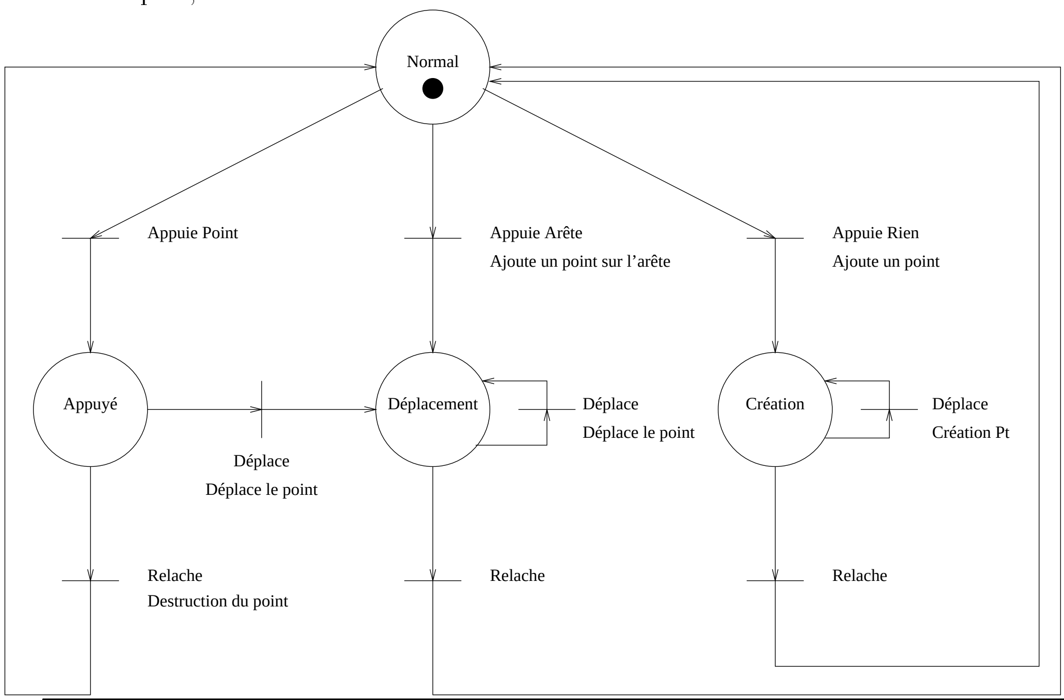
Une action liée à la transition est réalisée.

*Les réseaux de pétri sont facilement codable et vérifiable (Dead-Lock)*

### 3.1.1 Exemple : Editeur de polygone

Des graphes de Petri simplifiés (un seul jeton) sont courant en informatique, en effet l'état du graphe peut être codé par un seul entier : le numéro du noeud actif.

D'autre part, les conditions sont souvent des *événements*




| | Appuie sur sommet | Appuie sur arête | Appuie sur rien | Déplace | Relache |
|---|---|---|---|---|---|
| (N)ormal | (A) | (D) Crée pt sur arête | (C) Crée pt | (N) | (?) |
| (A)ppuyé | (?) | (?) | (?) | (D) Déplace pt | (N) Détruit pt |
| (D)éplacement | (?) | (?) | (?) | (D) Déplace pt | (N) |
| (C)réation | (?) | (?) | (?) | (C) Crée pt | (N) |

Dans les cas complexes, plutôt que de faire un double switch (lent) il suffit d'initialiser un tableau à 2 dimension contenant l'état suivant et un pointeur sur la fonction à exécuter.

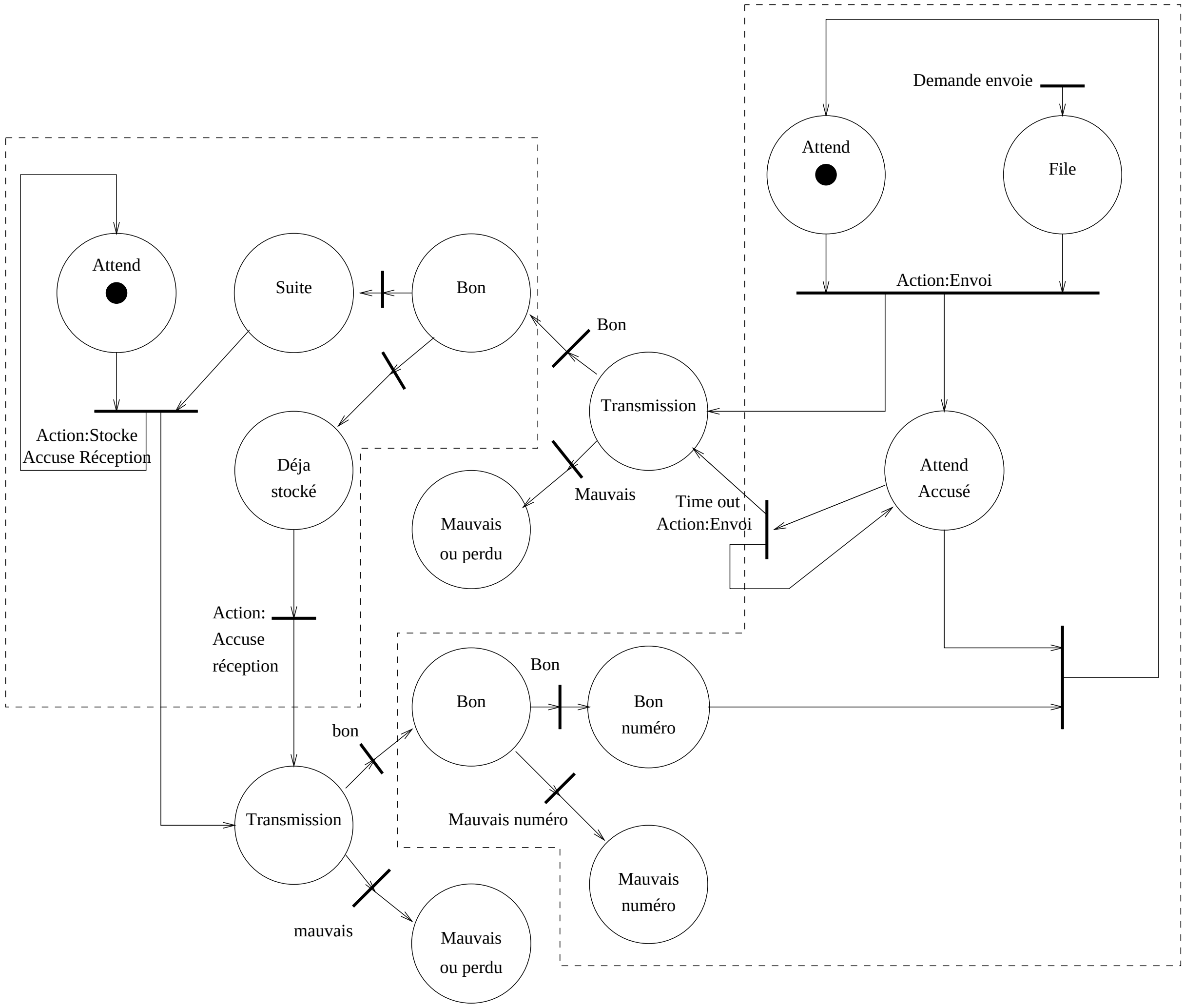
```
switch(événement)
   {
   case A :
      switch(état)
         {
         case 1: action ;
                 état = état_suivant ;
                 break ;
         ...
         }
      break ;
   ...
   }
```

```
struct { void action() ;
         int état_suivant ;
       } automate[][] = { {A, E}, ... } ;

automate[événement][état].action(événement, état) ;
état = automate[événement][état].état_suivant ;
```

### 3.1.2 Exemple : Communication monodirectionnelle

On suppose que la transmission se fait avec perte et changement d'ordre des paquets.



# 4 Communication client/serveur

Pour les communications entre machines, le standard est TCPIP.
La communication peut s'établir à un niveau plus ou moins élevé.

- ...
- Socket
- ...
- RPC
- ...
- DCE (Distributed Computing Environment) norme OSF

Les paramètres de la communication sont divers:
- Le protocole utilisé
- La confidentialité est importante
- La fiabilité permet de simplifier les programmes
- La compression permet d'augmenter le débit
- Une représentation uniforme des données

## 4.1 Protocole

Petit Larousse : *n.m. (gr* prôtokollon*, de prôtos, premier, et* kolla*, colle) Ensemble des règles établies en matière d'étiquette, d'honneur et de préséances dans les cérémonies officelles.*

On enlève «honneur» et cela convient parfaitement à l'informatique... En effet, si l'on veut qu'une communication se passe bien, il faut respecter certaines règles.

Attention chacun des acteurs informatiques interprète son rôle dans le protocole.

Les protocoles sont souvent complexes et doivent être *vérifié* pour être certain de leur fonctionnement. La moindre erreur conduit au bloquage.

Exemple de protocoles que vous connaissez déjà :
- La définition d'un langage de programmation est un protocole (mono directionnel) que vous devez suivre si vous voulez communiquer du travail à un compilateur de ce langage.
- Le distributeur de billet de banque obéit à un protocole bien défini et vous donne les instructions afin que vous suiviez le même protocole.
- Les machines communiquent entre elles en utilisant de nombreux protocoles de niveau logiques plus ou moins élevé : Ethernet, IP, TCP, ...
- Le navigateur WWW et le serveur obéissent au protocole HTTP afin de se communiquer les informations. Le protocole HTTP suppose que TCPIP fonctionne entre les deux machines.

Attributs du protocole :
- Synchrone/Asynchrone
- Lisible/Illisible
- Sans ou avec état courant

## 4.2 Socket

Un canal de communication fiable (mais non «securisé») peut être établi entre 2 sockets. Les sockets peuvent évidemment se trouver sur des machines différentes.

Le serveur est lancé dès la mise en route de la machine, il crée un socket en attente de connexion/données.

La connexion peut être *connection less* ou *stream*. Dans le cas d'un STREAM c'est équivalent à un *pipe*.

Le socket assure une liaison fiable. Elle détecte les arrêts de machines mais n'assure pas les reprises.

Les sockets font transiter des caractères. ATTENTION au binaire qui change d'une machine à l'autre.

Le socket est nommé par un numéro, lui-même nommé par `/etc/services`

Seul `root` peut lancer des serveur sur les port $< 1024$

### 4.2.1 Exemple

L'exemple d'utilisation suivant montre un serveur qui accepte plusieurs demandes de connexion simultanées et il les gère toutes lui-même. Il engage donc des dialogues simultanés. Les connexions ont une durée c'est une STREAM. On peut faire un dialogue "connection less". Le serveur s'occuppe de plusieurs sockets en même temps.

Dans la réalité, souvent (FTP, TELNET, ...) le serveur fait autant de `fork` que de clients. Les problèmes surgissent alors quand les différents fils du serveur doivent communiquer ensemble.

Les idées :

- Ouvrir un socket (`socket()`)
- Nommer un socket, pour un service (`bind()`)
- Dire au socket de se connecter a un serveur (`connect()`)
- Dire au socket d'attendre une connexion (`listen()`)
- Dire au socket d'accepter une connexion (`accept()`)

Utile : `select()`, `read()`, `write()`, `signal()`

# Utilitaires

```
/* -*- comment-column: 45 -*- */
#include "utilitaires.h"
#include <string.h>

/*
 *******************************************************************************
 * Ouverture d'un socket de type flot de donnees
 *******************************************************************************
 */

int ouvre_socket(int port)
{
  int s ;                                    /* Le socket que l'on ouvre     */
  struct sockaddr_in name ;                  /* Nom du socket                */
  int err ;                                  /* Numero d'erreur de commande  */

  s = socket(AF_INET ,                       /* Le socket est sur le reseau  */
             SOCK_STREAM,                    /* C'est analog a un pipe       */
             0                               /* IP (/etc/protocols)          */
             ) ;

  if ( s<0 )
    {
      perror(__FILE__ "/ouvre_socket_anonyme") ;
      fprintf(stderr, "Ne peut pas ouvrir le socket de connexion\n") ;
      exit(1) ;
    }

  /*
   * Maintenant on nomme le socket, et on dit avec qui il communique
   */

  memset(&name, 0, sizeof(name)) ;
  name.sin_family      = AF_INET;            /* Sur le reseau                */
  name.sin_addr.s_addr = INADDR_ANY ;        /* N'importe qu'elle machine    */
  name.sin_port        = htons(port) ;       /* Mon numero de port           */
                                             /* (Suis pas sur du htons.....) */
  err = bind(s,                              /* Le socket en question        */
             (struct sockaddr*)&name,        /* Son nouveau nom              */
             sizeof(name)
             ) ;

  if ( err != 0 )
    {
      perror(__FILE__ "/ouvre_socket_anonyme") ;
      fprintf(stderr,
              "Ne peut donner de nom au socket de connexion\n"
              "Attendez environ une minute avant de relancer le programme\n");
      close(s) ;
      exit(1) ;
    }

  return(s) ;
}

/*
 *******************************************************************************
 * Attente evenement sur plusieur fichiers en meme temps
 *
 *******************************************************************************
 */

static fd_set attente = { { 0 } } ;          /* Les fildes a attendre       */

void ajoute_fildes(int fildes)
{
  if ( fildes>=MAX_FILDES
       || fildes<0
       || FD_ISSET(fildes, &attente)
       )
    {
      fprintf(stderr,
              "%s:%d ajoute_fildes(%d) numero invalide ou deja actif\n",
              __FILE__, __LINE__, fildes) ;
      exit(1) ;
    }
  FD_SET(fildes, &attente) ;
}

void enleve_fildes(int fildes)
{
  if ( fildes>=MAX_FILDES
       || fildes<0
       || !FD_ISSET(fildes, &attente)
       )
    {
      fprintf(stderr, "%s:%d enleve_fildes(%d) numero invalide ou inactif\n",
              __FILE__, __LINE__, fildes) ;
      exit(1) ;
    }
  FD_CLR(fildes, &attente) ;
}
```

```
int attente_fildes()
{
  fd_set r, x ;                              /* En lecture et special       */
  int max ;                                  /* Numero du plus grand fildes */
  int err ;
  int i ;
  /*
   * Preparation des parametres
   */
  max = 0 ;
  fprintf(stderr, "Attente sur :") ;
  for(i=0;i<MAX_FILDES;i++)
    if (FD_ISSET(i, &attente))
      {
        max = i ;
        fprintf(stderr, " %d", i) ;
      }
  fprintf(stderr, "\t") ;
  x = r = attente ;
  /*
   * On va attendre reellement
   */
  err = select( MAX_FILDES, &r, NULL, &x, 0) ;  /* Attente             */
  /*
   * Recherche du fichier declenchant
   */
  for(i=0;i<max+1;i++)
    if (FD_ISSET(i,&r) || FD_ISSET(i,&x))
      return(i) ;

  return(-1) ;
}

/*
 ******************************************************************************
 * Ouvre une connexion vers un serveur
 ******************************************************************************
 */

int ouvre_connexion(char *machine, int port)
{
  int s ;                                    /* Le socket que l'on ouvre     */
  struct sockaddr_in name ;                  /* Nom du socket                */
  int err ;                                  /* Numero d'erreur de commande  */
  struct hostent *mmserv ;                   /* Le host du serveur           */

  s = socket(AF_INET ,                       /* Le socket est IP             */
             SOCK_STREAM,                    /* C'est analog a un pipe       */
             0                               /* Protocol IP "/etc/protocols" */
             ) ;

  if ( s<0 )
    {
      perror(__FILE__ "/ouvre_connexion") ;
      fprintf(stderr, "Ne peut pas ouvrir le socket de connexion\n") ;
      exit(1) ;
    }
  /*
   * On dit a qui l'on veut connecter le socket
   */

  mmserv = gethostbyname(machine) ;
  if ( mmserv==0 )
    {
      perror(__FILE__) ;
      fprintf(stderr, "Ne trouve pas la machine serveur\n" );
      fprintf(stderr, "Verifier le nom de la machine\n" );
      exit(1) ;
    }
  memset(&name, 0, sizeof(name)) ;
  memcpy(
         (char*)(&(name.sin_addr.s_addr)) ,
         mmserv->h_addr ,
         mmserv->h_length
         ) ;
  name.sin_family      = AF_INET;            /* Sur le reseau                */
  name.sin_port = htons(port) ;

  err = connect(s,
                (struct sockaddr*)&name,
                sizeof(name)
                ) ;

  if ( err<0 )
    {
      perror(__FILE__) ;
      fprintf(stderr, "La connexion ne se fait pas\n" );
      fprintf(stderr, "Le serveur ne doit pas marcher\n" );
      exit(1) ;
    }
  return(s) ;
}

/*
 * La fonction "accept" retourne des informations
```

```
 * sur la machine qui se connecte.
 * Pour vous simplifier la vie, je le cache
 */

int accepte_connexion(int socket)
{
  int i ;
  struct sockaddr s ;

  i = sizeof(s) ;
  return(accept(socket,&s,&i)) ;
}
```

# Client

```
/* -*- comment-column: 45 -*- */
#include "utilitaires.h"
#include <string.h>

/*
 * Un exemple de client
 *
 * Utilisations : NOTEZ BIEN LA MANIERE DONT LA COMMANDE SUIVANTE EST TAPEE
 * en effet, le parametre entre guillemet contient 2 returns
 * Lez "$" et ">" representent le prompt.
 *
 * $ client bat710 80 "GET /Welcome.html HTTP/1.0
 * >
 * > "
 *
 * Autre utilisation, avec le serveur dont vous avez les sources
 * (n'oubliez pas de lancer le serveur)
 *
 * $ client MACHINE 6001 "echo coucou"
 *
 * Ou encore mieux, une interogation parallele de serveurs
 *
 * $ client MACHINE 6001 "echo coucou" MACHINE 6002 "echo toto"
 */

int main(int argc, char **argv)
{
  int s[MAX_FILDES] ;                        /* Les sockets a lire */
  int so ;                                   /* Le socket a lire */
  char buffer[BUF] ;                         /* Buffer de lecture */
  int nb ;                                   /* Nombre de caracteres lues */
  int nb_connexions ;                        /* Nombre de connexions */
  char *nom_machine, *numero_port, *message ;
  int i ;

  if ( argc < 4 )
    {
      fprintf(stderr,
              "Pour chacun des serveurs a interoger, il faut donner\n"
              "trois parametres :\n"
              "  - nom de machine\n"
              "  - numero de port\n"
              "  - message a envoyer\n"
              ) ;
      exit(1) ;
    }
  /*
   * On ouvre chaque socket et on envoi le message
   */
  nb_connexions = argc/3 ;
  for(i=0;i<nb_connexions;i++)
    {
      nom_machine = argv[i*3+1] ;
      numero_port = argv[i*3+2] ;
      message     = argv[i*3+3] ;
      s[i] = ouvre_connexion(nom_machine, atoi(numero_port) ) ;
      if( write(s[i], message, strlen(message)) != strlen(message) )
        {
          fprintf(stderr, "Cas non prevu, il faudrait ecrire la suite\n");
          exit(1) ;
        }
      ajoute_fildes(s[i]) ;
    }
  /*
   * Tant qu'il y a des connexions, on attent une info du serveur
   */
  while( nb_connexions )
    {
      so = attente_fildes() ;
      nb = read(so,buffer,BUF) ;             /* On lit des donnees */
      if ( nb <= 0 )
        {
          /*
           * Le socket a ete ferme par le serveur
           */
          fprintf(stderr, "[%d]fermeture\n", so) ;
          close(so) ;                        /* On libere le fichier */
          enleve_fildes(so) ;                /* On attend plus dessus */
          nb_connexions-- ;
        }
      else
        {
          /*
           * On affiche ce qui a ete lu.
           * ATTENTION : une transmission peut etre decoupee
           * en plusieur morceau meme si elle a ete envoye
           * en une seule fois
           */
          buffer[nb] = '\0' ;   /* Pour pouvoir utiliser "printf" */
          fprintf(stderr, "[%d]%s", so, buffer) ;
        }
    }
  return(0) ;
```

```
}
```

# Serveur

```c
/* -*- comment-column: 45 -*- */
#include "utilitaires.h"
#include <signal.h>
#include <string.h>

/*
 * Un exemple de serveur, mono-tache mais acceptant plusieur connexions.
 *
 * Utilisation :
 *   - Lancer le serveur et le laisser tourner dans une fenetre
 *     en tapant "serveur 6001"
 *   - Faite un "telnet MACHINE_DU_SERVEUR 6001"
 *   - Tapez du texte, le serveur l'affichera
 *   - Si vous tapez un texte commencant par "echo " alors
 *     en plus, il vous retournera le texte que vous avez tape.
 *   - Vous pouvez lancer plusieur "telnet" venant de differentes machines
 *     sur le serveur et taper des textes dans les differents "telnet"
 *
 * Pour sortir de "telnet" tapez : CONTROL-] puis "quit"
 * Le serveur vous indiquera la fermeture de la connexion.
 *
 * Essayez : "netscape http://MACHINE_DU_SERVEUR:6001/toto" !!!
 *
 * Pour avoir + d'infos sur les serveurs avec >10000 client :
 * http://www.kegel.com/c10k.html
 */

int main(int argc, char **argv)
{
  int s ;                                    /* Le socket a lire */
  int msocket ;                              /* Le socket de connexion */
  char buffer[BUF] ;                         /* Buffer de lecture */
  int nb ;                                   /* Nombre de caracteres lues */

  if ( argc != 2 )
    {
      fprintf(stderr, "Il faut donner le numero du port >10000\n") ;
      exit(1) ;
    }

  signal(SIGPIPE, SIG_IGN) ;                 /* Ecriture sans lecteur */
  msocket = ouvre_socket(atoi(argv[1])) ;    /* Ouverture du socket */
  listen(msocket, 1) ;                       /* Attente de connexion */
  ajoute_fildes(msocket) ;                   /* On attendra dessus */

  for(;;)
    {
      s = attente_fildes() ;                 /* On attend une intervention */

      if ( s == msocket )                    /* Demande de connexion ? */
        {
          s = accepte_connexion(msocket) ;   /* On accepte la connexion */
          ajoute_fildes(s) ;                 /* Attente sur le nouveau socket*/
          fprintf(stderr, "[%d]ouverture\n", s) ;
        }
      else
        {
          nb = read(s,buffer,BUF) ;          /* On lit des donnees */
          if ( nb <= 0 )
            {
              /*
               * Le socket a ete ferme par le client
               */
              fprintf(stderr, "[%d]fermeture\n", s) ;
              close(s) ;                     /* On libere le fichier */
              enleve_fildes(s) ;             /* On attend plus dessus */
            }
          else
            {
              /*
               * On affiche ce qui a ete lu.
               * ATTENTION : une transmission peut etre decoupee
               * en plusieur morceau meme si elle a ete envoye
               * en une seule fois
               */
              buffer[nb] = '\0' ;       /* Pour pouvoir utiliser "printf" */
              fprintf(stderr, "[%d]%s", s, buffer) ;
              if ( strncmp(buffer, "echo ", 5) == 0 )
                if ( write(s, buffer+5, nb-5) != nb-5 )
                  {
                    fprintf(stderr,
                            "Cas non prevu,"
                            " il faudrait ecrire la suite\n");
                    exit(1) ;
                  }
            }
        }
    }
}
```

### 4.2.2 Inetd

Il est un peut lourd d'avoir autant de processus actif que de services, généralement il y a beaucoup de services rarement actif.

Pour "résoudre" ce problème, un processus nommé `inetd` se met en attente d'information sur TOUS les *sockets*. Lorsqu'une information arrive, il active le bon programme, et le laisse actif ou non.

Il gère en plus les problèmes d'interdiction d'accès à certaines machines.

Une partie du fichier de configuration (`/etc/inetd.conf /etc/inetd.sec`)

```
#       ARPA/Berkeley services
ftp          stream tcp nowait root /etc/ftpd     ftpd -l
login        stream tcp nowait root /etc/rlogind  rlogind
shell        stream tcp nowait root /etc/remshd   remshd
#
time         stream tcp nowait root internal
time         dgram  udp nowait root internal
#       rpc services, registered by inetd with portmap
#       Do not uncomment these unless your system is running portmap!
rpc  stream tcp  nowait  root  /usr/etc/rpc.rexd     100017  1    rpc.rexd
rpc  dgram  udp  wait    root  /usr/etc/rpc.rstatd   100001  1-3  rpc.rstatd
```

Si l'on utilise `inetd` on n'a pas besoin d'accepter la connexion. On travail directement avec l'entrée standard et la sortie standard.

Autre avantage, si le serveur plante, il sera relancé.

Le numéros de ports sont dans `/etc/services`

```
ftp-data      20/tcp                  # File Transfer Protocol (Data)
ftp           21/tcp                  # File Transfer Protocol (Control)
time          37/tcp  timeserver      # Time
time          37/udp  timeserver      #
```

Pour voir la liste des connexions: `netstat -na`

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        (state)
tcp        0      0  134.214.88.42.1782     134.214.88.54.6000     ESTABLISHED
tcp        0      0  134.214.88.42.1588     36.22.0.146.20         ESTABLISHED
tcp        0      0  134.214.88.42.1561     36.22.0.146.21         ESTABLISHED
tcp        0      0  *.1107                 *.*                    LISTEN
tcp        0      0  *.6000                 *.*                    LISTEN
tcp        0      0  134.214.88.42.2298     134.214.88.54.6000     ESTABLISHED
udp        0      0  *.2394                 *.*
udp        0      0  134.214.88.42.53       *.*
udp        0      0  127.0.0.1.53           *.*
```

## 4.3 RPC: Remote Procedure Call

Propriétés :

- Sans état.
- Transcodage de format (XDR eXternal Data Representation)
- Gestion de versions.
- Options de sécurité.

Pour des architectures complexe on utilise `rpcgen` pour générer les squelettes des applications clientes et serveuses. Ou les UIL de DCErpc.

Exemple simple:

### 4.3.1 Client

```
/* -*- comment-column: 45 -*- */
#include "rpc_util.h"

/*
 * Le client RPC
 */
int main(int argc, char **argv)
{
  struct mon_type parametre ;                /* Le parametre a envoyer */
  float resultat ;                           /* Le resultat du calcul */
  char nom_client[100] ;
  int err ;

  if (argc < 3)
    {
      fprintf (stderr, "usage: rpc_clie nom_de_machine nombre_flottant\n") ;
      exit (1) ;
    }

  /*
   * La valeur a elever a la puissance
   */
  parametre.valeur = atof(argv[2]) ;

  /*
   * Mon identite
   */
  gethostname( nom_client, sizeof(nom_client) ) ;
  parametre.nom_client = nom_client ;
  /*
   * Si resultat etait un "char *" et que l'on veuille
   * une allocation automatique, il faudrait indiquer ici :
   *    resultat = NULL ;
   */
  /*
   * On va appeler le service pour le demander le resultat avec
   * differentes puissances.
   */
  for( parametre.puissance = 1 ;
       parametre.puissance < 25 ;
       parametre.puissance++ )
    {
      err = callrpc(argv[1],                 /* Machine a interroger */
                    PROG_ESSAI,              /* Programme a interroger */
                    PROG_ESSAI_VERSION,      /* Version a appeler */
                    PROG_ESSAI_PUISSANCE,    /* Fonction a appeler */
                    (xdrproc_t)xdr_mon_type, /* Type du parametres en entree */
                    (char*)&parametre,       /* Valeur du parametre */
                    (xdrproc_t)xdr_float,    /* Type de la valeur retournee */
                    (char*)&resultat         /* Valeur retournee */
                    ) ;
      if ( err )
        {
          clnt_perrno(err) ;
          fprintf(stderr,"\nAu revoir\n") ;
          exit(1) ;
```

```
        }
      else
        {
          printf("Resultat : %g^%lu vaut %g en utilisant %s\n",
                 parametre.valeur, parametre.puissance, resultat, argv[1]) ;
          /*
           * Ce qui suit sert a liberer la memoire allouee par la valeur
           * retournee. Dans le cas du float cela ne sert a rien
           * car rien n'a ete alloue.
           *
           * Si la valeur retournee etait "xdr_mon_type" la chaine
           * de caracteres "nom_client" aurait ete liberee.
           * Apres la liberation 'dun pointeur, sa valeur devient NULL
           */
          xdr_free( (xdrproc_t)xdr_float, (char*)&resultat ) ;
        }
    }
  return(0) ;
}
```

### 4.3.2 Serveur

```
/* -*- comment-column: 45 -*- */

#include "rpc_util.h"

/*
 * La fonction d'élévation à la puissance,
 * c'est le service fourni par le serveur.
 *
 * ATTENTION : on retourne toujours un pointeur vers la données
 * à retourner même si c'est un simple flottant.
 *
 * N'oubliez pas que l'on ne peut pas retourner un pointeur sur une variable
 * locale (sauf si elle est static), sinon le programme fonctionne
 * aléatoirement.
 *
 * ATTENTION : en C, si vous declarez : "char toto[100] ;"
 * La valeur "toto" est egale a la valeur "&toto", on ne peut
 * donc pas prendre l'adresse d'un tableau,
 * il faut donc passer par un pointeur.
 */

char    *puissance(const struct mon_type* input)
{
  static float resultat ;
  int i ;

  fprintf(stderr, "input->valeur=%g  input->puissance=%lu client=%s\n",
          input->valeur, input->puissance, input->nom_client) ;
  resultat = input->valeur ;
  for( i=1 ; i<input->puissance ; i++ )
    resultat *= input->valeur ;

  return ((char *)&resultat) ;
}

/*
 * Ce serveur n'enregistre qu'un seul service.
 * Il pourrait tres bien en diffuser plusieurs
 * en appelant plusieur fois "registerrpc"
 */

int main ()
{

  registerrpc (PROG_ESSAI, PROG_ESSAI_VERSION,
               PROG_ESSAI_PUISSANCE,        /* Numero de la fonction */
               puissance,                   /* Pointeur sur la fonction */
               xdr_mon_type,                /* Parametre en entree */
               xdr_float                    /* Parametre en sortie */
               ) ;

  svc_run () ;                              /* Ne revient jamais */

  fprintf(stderr, "Error: svc_run returned!\n") ;
  return(1) ;
}
```

#### 4.3.3 XDR

```
/* -*- comment-column: 45 -*- */
#ifndef RPC_UTIL_H
#define RPC_UTIL_H
#include        <stdio.h>
#include        <stdlib.h>
#include        <unistd.h>
#include        <rpc/rpc.h>
#define PROG_ESSAI           1234560         /* Numero de serveur */
#define PROG_ESSAI_VERSION   VOTRE_UID       /* Version du serveur */
                                             /* On prend l'UID pour eviter */
                                             /* les conflits en TP */
#define PROG_ESSAI_PUISSANCE 1               /* Numero de la fonction */
/*
 * Pour communique avec le serveur, j'ai besoin de lui passer des parametres.
 * Avec les RPC on ne peut passer qu'un seul parametre.
 * Il faut donc passer une structure.
 *
 * Pour elever un nombre a une certaine puissance,
 * j'ai besoin du nombre et de la puissance.
 *
 */
struct mon_type
{
  float   valeur ;
  u_long  puissance ;
  char    *nom_client ;
} ;

/*
 * Représentation XDR de la structure ci-dessus.
 *
 * Cette fonction est reversible :
 * elle peut traduire du format XDR vers ma structure ou l'inverse.
 *
 * Elle est aussi utilisée pour libérer la mémoire.
 *
 */
bool_t xdr_mon_type(XDR *xdrs, struct mon_type *t)
{
  bool_t r ;                                 /* Erreur de conversion ? */

  r  = xdr_float  (xdrs, &t->valeur ) ;
  r &= xdr_u_long (xdrs, &t->puissance ) ;
  r &= xdr_wrapstring (xdrs, &t->nom_client) ; /* Il fait un malloc si neces.*/

  return( r ) ;
}
/*
 * Sous Linux le prototype n'existe pas.
 * On en met un bidon ici.
 */

int registerrpc() ;

/*
 * ATTENTION DANGER, il ne faut pas faire :
 * return( xdr_u_float(xdrs,&t->valeur) & xdr_u_long (xdrs, &t->puissance) ) ;
 * car ceci n'est pas portable (ordre d'evaluation aleatoire)
 */
/*
 * On peut evidemment mettre des tableaux, chaine, ...
 * Si l'on ne veut pas de traduction, on peut utiliser :
```

```
 * "xdr_opaque" si le type est IDENTIQUE sur TOUTE les machines
 */
#endif
```

### 4.3.4 Portmap

`portmap` est un processus enregistrant les `registerrpc` et assignant un numéro de port au serveur. Tout client commence donc par dialoguer avec `portmap`

La commande `rpcinfo -p` :

```
program vers proto   port
 100007    2   tcp    734  ypbind
 100007    2   udp    736  ypbind
 100007    1   tcp    734  ypbind
 100007    1   udp    736  ypbind
 100024    1   udp    750  status
 100024    1   tcp    752  status
```

## 4.4 DCE : Distributed Computing Environment

`http://www.osf.org/dce/index.html`

DCE a été créé afin de faciliter la création d'application client/serveur. Il offre des langages de haut niveaux (IDL) permettant de définir des applications client/serveur très facilement en garantissant le fiabilité et la sécurité.

Il offre d'autres part des applications de base toutes faites pour faciliter l'administration de réseaux de machines. Elles remplacent en fait des applications faites au coup par coup par des gens différent. On a donc avec DCE un environnement complet et LOGIQUE.

Enormément de chose deviennent transparente avec DCE. Par exemple, on peut utiliser pour accéder à des services un nom logique qui ne dépend pas de la machine sur lequel tourne le serveur.

DCE fonctionne maintenant (au moins en tant que client) dans presque tous les environnements. Son utilisation n'a réellement commencé qu'en 1995, c'est normalement le système de l'avenir.

*DCE consists of multiple components which have been integrated to work closely together. They are the Remote Procedure Call (RPC), the Cell and Global Directory Services (CDS and GDS), the Security Service, DCE Threads, Distributed Time Service (DTS),and Distributed File Service (DFS). The Threads, RPC, CDS, Security, and DTS components are commonly referred to as the "secure core" and are the required components of any DCE installation. DFS is an optional component. DCE also includes administration tools to manage these components.*

# 5 Exemples, 1 serveur, synchrone

## 5.1 telnet RFC 854

*Telnet uses a single TCP connection, and its normal data stream ("Network Virtual Terminal" or "NVT" mode) is 8-bit ASCII with escape sequences to embed control functions. Telnet also allows the negotiation of many optional modes and functions.*

*The primary Telnet specification is to be found in RFC-854 [TELNET:1], while the options are defined in many other RFCs; see Section 7 for references.*

`telnet NOM` permet de "se connecter" sur une autre machine en entrant en dialogue avec un autre programme (pas forcement un login)

En plus des caractères dans les 2 sens, il faut transporter d'autre informations, taille écran, echo, ...

C'est un protocol symétrique, d'où sa complexité. En fait c'est une architecture client/client!

On traite le code `0xFF` comme un caractère de contrôle. Pour envoyer ce caractère, il faut le doubler.

Ce qui suit ce caractère est une chaîne plus ou moins longue.

Le dialogue est fait pour être très souple, évolutif et permettre a des versions différentes de fonctionner.

Les options sont au départ toute "désactivées".

Les options sont ON ou OFF indépendemment de chaque coté.

La base du dialogue est :

- WILL
  Je l'envoie : J'aimerais faire XXX (j'attend DO ou DON'T)
  Je le reçoie : Il aimerais faire XXX (possible réponse à DO)
- DO
  Je l'envoie : J'aimerais que tu fasse XXX (j'attend WILL ou WON'T)
  Je le reçoie : Il aimerais que je fasse XXX (possible réponse à WILL)
- WON'T
  Je l'envoie : Je ne fais plus XXX (possible réponse à DO)
  Je le reçoie : Il ne fait plus XXX
- DON'T
  Je l'envoie : Ne fais pas XXX (possible réponse à WILL)
  Je le reçoie : Je ne fais plus XXX

Le protocol est complexe car il est complètement asynchrone.

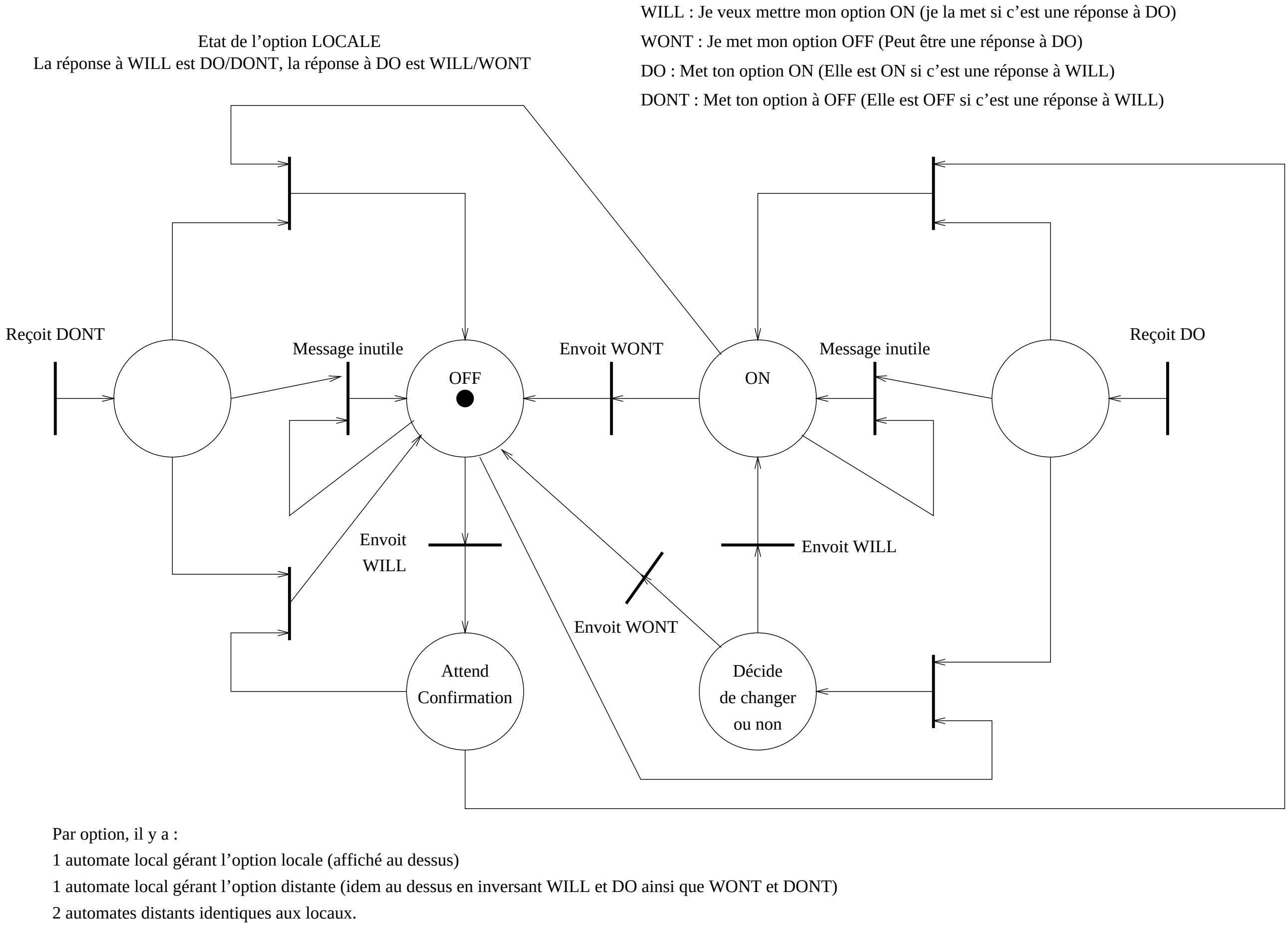
```
Connected to dessiipi.univ-lyon1.fr.
Escape character is '^]'.
SENT do SUPPRESS GO AHEAD                          # Demande SUPPRESS GO AHEAD
SENT will TERMINAL TYPE (don't reply)              # On supporte TERMINAL TYPE
SENT will NAWS (don't reply)                       # On supporte NAWS
< 0x0   fffd24
> 0x0   fffd03fffb18fffb1f
RCVD do 36 (reply)                                 # On me demande 36
SENT wont 36 (don't reply)                         # Je ne le fait pas :-)
> 0x0   fffc24
< 0x0   fffd18
RCVD do TERMINAL TYPE (don't reply)                # On me demande TERM TYPE
< 0x0   fffb03fffd1ffffa1801fff0
RCVD will SUPPRESS GO AHEAD (don't reply)          # Il accepte SUPRESS GO AHE
RCVD do NAWS (don't reply)                         # On me demande NAWS
Sent suboption NAWS 0 80 (80) 0 24 (24)            # On envoit la taille
Received suboption Terminal type - request to send.
> 0x0   fffa1f00500018fff0fffa180048505445524dfff0
< 0x0   fffb01fffd010d0a48502d5558206465737369697069204 12e30392e30352041
< 0x20  20393030302f3731322028747479735290d0a0d0a
RCVD will ECHO (reply)                             # Distant fait l'ECHO?
SENT do ECHO (reply)                               # Je lui dit Oui
RCVD do ECHO (reply)                               # Il me demande de faire EC
SENT wont ECHO (don't reply)                       # Je le fais pas
> 0x0   fffd01fffc01

HP-UX dessiipi A.09.05 A 9000/712 (ttys5)
```

La commande telnet permet d'accéder à tous les socket!

```
telnet miage 13
```

Implémentation: il y a autant de `telnetd` que de sessions de `telnet`.




## 5.2 FTP

Pour «voir» une partie du protocol:

`telnet miage 21`

Vous pouvez alors taper des commandes en clair. Mas vous ne pouvez pas transferer de fichiers. Les transferts se font avec un autre socket. Pour le transfert des commande, le protocol TELNET est respecté, l'echo est fait en local.

La commande `ftp` est plus riche car elle construit ces commandes en utilisant les commandes de base.

Il y a autant de serveurs que de connexions.

Le protocol est complexe car ancien:

- Les octets transférés peuvent être sur 7 ou 8 bits
- Les fichiers peuvent être: Ascii, Binaire, Enregistrements, Troués.
- La transmission peut être avec reprise, compressé, ...

Le problème de FTP était que les machines étaient complètement hétéroclites à l'époque et qu'il fallait faire des conversions tordues pour passer un fichier d'une machine à l'autre.

La difficulté de FTP est l'établissement du canal des données.

Le fonctionnement est le suivant : le client cré un socket, demande le `bind` sur une adresse quelconque, se met en `listen` et envoi le numéro du port au serveur FTP, qui lui va se connecter au client!

## 5.3 HTTP :

Documentation complète dans: `http://www.w3.org/hypertext/WWW/Protocols/HTTP/HTTP2.html`

Les messages HTTP sont en clair. Le protocol est évolutif.

Pour éliminer tout risque de confusion le texte des données qui circulent en direction du serveur sont *escapée*. Le blanc par exemple car c'est un séparateur.

### 5.3.1 HTTP 0.9

Pour essayer un requête simple, faite `telnet www710 80` et tapez: `GET /Welcome.html`

Vous recevez alors directement le fichier que vous avez demandé.

### 5.3.2 HTTP 1.0

En faites requêtes actuelles suivent un protocol plus compliqué.

Voici par exemple une connexion `Mosaic`

```
GET /Welcome.html HTTP/1.0
Accept: application/html
Accept: application/pgp
Accept: application/postscript
Accept: application/x-dvi
...
Accept: audio/*
Accept: audio/x-aiff
...
Accept: image/*
Accept: image/gif
Accept: image/jpeg
...
Accept: object/x-oogl
Accept: text/html
Accept: text/plain
Accept: text/richtext
...
Accept: video/*
Accept: video/mpeg
Accept: */*
Connection: Keep-Alive
User-Agent:  NCSA_Mosaic/2.7b1 (X11;HP-UX A.09.05) libwww/2.12 modified
Extension: Notify-Domain-Restriction
```

La requête dépend de votre fichier `.mailcap`

Exemple avec `netscape`

```
GET /Welcome.html HTTP/1.0
User-Agent: Mozilla/1.1N (X11; I; HP-UX A.09.05 9000/715)
Accept: */*
Accept: image/gif
Accept: image/x-xbitmap
Accept: image/jpeg
```

Les fichiers demandés sont envoyés sous la forme d'un document au format MIME. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies

Voici une entête MIME pour un document HTML

```
HTTP/1.0 200 Document follows
MIME-Version: 1.0
Server: CERN/3.0pre5
Date: Wednesday, 13-Dec-95 11:32:59 GMT
Content-Type: text/html
Content-Length: 2651
Last-Modified: Friday, 31-Mar-95 07:46:24 GMT

texte HTML
```

Evidemment d'autre méthodes que GET existent. Cela afin de modifier les fichiers, les blocker, les protéger, ...

### 5.3.3 Ce qui est transmis pour WWW

Des fichiers au format HTML (Hot Text Markup Language) qui est une instanciation de SGML (norme ISO: Simple Generic Markup Language) Evidemment des images, son, fichiers de données.

CE QUI EST TRANSMIS NE DOIT PAS PERMETTRE DE FAIRE DES CHEVAUX DE TROIS.

Le futur est le langage JAVA qui peut être compilé dans un code indépendant du processeur et qui peut être interprété sans danger d'introduire des chevaux de trois. Ce langage va permettre de faire évoluer WWW dans la direction que l'on veut sans avoir besoin d'améliorer les browsers.

## 5.4 SNMP : Simple Network Management Protocol

Avec SNMP, tout objet connecté sur le réseau fait fonctionner un serveur (sous UNIX `snmpd`) qui répond à des requêtes GET ou SET de variables.

La communication avec SNMP est *connection less*

Lors de la connection on doit fournir un nom et un mot de passe. Le nom est généralement par défaut: `public` et le mot de passe sert principalement à autoriser des requêtes modifiant l'état de la machine.

Les variables sont nommées par une liste d'entier. Ce que l'on appel des MIB sont des bases de données qui associent à des noms en clair le numéro (liste entier), le type et d'autres informations. Vous pouvez trouver des exemples de MIB dans `/vol/local.hp700/lib/mibdir`

```
org                  iso.3
dod                  org.6
internet             dod.1
mgmt                 internet.2
mib-2                mgmt.1
directory            internet.1
experimental         internet.3
private              internet.4
enterprises          private.1
hp                   enterprises.11
nm                   hp.2
hpsystem             nm.3
general              hpsystem.1
computerSystem       general.1
computerSystemUsers  computerSystem.2   Gauge     read-only     mandatory
```

`computerSystemUsers` a comme *nom* : 1.3.6.1.2.1.1.3.4.1.11.2.3.1.1.2 C'est un entier que l'on peut seulement accéder en lecture.

Les types peuvent entre autre être entier, chaîne, table ou structures.

On peut aussi demander au demon d'envoyer des messages d'alarmes en cas de dépassement de certaines valeurs. Disque plein, trop d'erreur réseau, plus de toner.

### 5.4.1 Exemples

Les commandes "snmp-..." fonctionnent sur HP. Ce sont des utilitaires gratuit (`tricklet`)

Pour récupérer un valeur particulière:

`echo "computerSystemUpTime" | snmp-get miage`

*computerSystemUpTime[0]=58852500*

`echo "gdStatusTonerLow" | snmp-get ligia600`

*gdStatusTonerLow[0]=0*

Récupère des informations sur la table des systèmes (1 seul système dans l'exemple)

`echo "computerSystem[]" | snmp-table miage`

```
computerSystemUpTime[0]=58863600;
computerSystemUsers[0]=9;
computerSystemAvgJobs1[0]=151;
computerSystemAvgJobs5[0]=212;
computerSystemAvgJobs15[0]=189;
```

Pour voir les systèmes de fichiers montés (2 dans l'exemple):

`echo "fileSystemEntry[]" | snmp-table bdhp1`

```
fileSystemID1[119542784.0]=119542784
fileSystemID1[119543296.0]=119543296
fileSystemID2[119542784.0]=0
fileSystemID2[119543296.0]=0
fileSystemName[119542784.0]="/dev/dsk/c201d4s0"
fileSystemName[119543296.0]="/dev/dsk/c201d6s0"
fileSystemBlock[119542784.0]=510735
fileSystemBlock[119543296.0]=412884
fileSystemBfree[119542784.0]=153289
fileSystemBfree[119543296.0]=205669
fileSystemBavail[119542784.0]=153289
fileSystemBavail[119543296.0]=164380
fileSystemBsize[119542784.0]=4096
fileSystemBsize[119543296.0]=1024
fileSystemFiles[119542784.0]=335872
```

```
fileSystemFiles[119543296.0]=202368
fileSystemFfree[119542784.0]=265539
fileSystemFfree[119543296.0]=189223
fileSystemDir[119542784.0]="/disc2"
fileSystemDir[119543296.0]="/"
```

Pour voir le traffic réseau :

`echo "ieee8023MacEntry[]" | snmp-table mibri1`

```
ieee8023MacIndex[1]=1
ieee8023MacIndex[2]=2
ieee8023MacTransmitted[1]=82725415
ieee8023MacTransmitted[2]=91869649
ieee8023MacNotTransmitted[1]=0
ieee8023MacNotTransmitted[2]=0
ieee8023MacDeferred[1]=60727
ieee8023MacDeferred[2]=136164
ieee8023MacCollisions[1]=978942
ieee8023MacCollisions[2]=2675279
ieee8023MacSingleCollisions[1]=258202
...
```

## 5.5 NFS: Network File System

Ce qui est requis: non sujet aux pannes, en fait le plus fiable possible. Portable, indépendant de la plateforme.

Le choix le plus important est le fait que la communication est connexion-less. Ceci veut dire que toutes les opérations sont atomiques et donc chaque opération est fiable. Entre chaque opération, si le serveur défaille, ce n'est pas grave. Les opérations NFS sont blocantes tant qu'elles ne sont pas terminées.

La communication est faite à l'aide de RPC.

Rien n'est crypté par NFS.

### 5.5.1 Coté serveur

Serveur de RPC classique avec vérification du droit d'utilisation à *chaque demande*.

Plusieur serveurs lancés en même temps car évidemment chaque instruction de lecture ne répond pas immédiatement.

Avec UNIX, ce sont les processus `nfsd`

Le processus `mountd` autorise ou non l'autre machine à utiliser le disque local (`/etc/exports` et la commande `exportfs`).

Remarque: il est faisable (et fait) d'avoir des serveurs NFS ne diffusant pas des disques locaux mais d'autres choses, dans la plupart des cas un changement de protocol ou de format de données: AUTOMOUNT, FTP, HTTP, APPLESHARE, SQL, Cryptage, Compression, ...

On peut installer un serveur NFS sans avoir à modifier le noyau du système.

### 5.5.2 Coté client

Ici, c'est un peu plus compliqué.

Il faut utiliser la commande `mount` pour indiquer au système qu'une hiérarchie de fichiers n'est plus dans l'environnement courant. On lui indique alors sur quel périphérique ou machine les fichiers sont accessibles. Ceci est standard UNIX. La commande `umount` permet de démonter.

Comme c'est le système qui gère cela, dans une architecture monolithique, il faut modifier le noyau afin qu'il fasse des requêtes (RPC) au serveur NFS.

### 5.5.3 Example

Quelqu'un tape: `ls /net/bat710/disc0/ftp`

Etape 1 : `/` c'est le disque racine bien connu par le système.

Etape 2 : `/net` c'est un disque `mounted` avec le protocol NFS. Le système va donc interroger la serveur NFS en lui donnant la suite du chemin.

Etape 3 : `/net/bat710` le serveur NFS est dans ce cas SUR LA MEME MACHINE et est en fait le processus `automount`. Ce serveur sert à faire des `mount` et `unmount` sans intervention de `root`. Dans cet exemple, il s'aperçoit que l'on veut accéder aux disques sur la machine nommée `bat710`. Il fait donc les montages nécessaires.

Etape 4 : `/tmp_mnt/net/bat710/disc0` est monté.
Un lien symbolique de `/net/bat710/disc0` vers `/tmp_mnt/net/bat710/disc0` est créé.

Etape 5 : Le système s'aperçoit que `disc0` est un lien symbolique et le suit donc.

Etape 6 : `/tmp_mnt` directory normale

Etape 7 : `/tmp_mnt/net` idem

Etape 8 : `/tmp_mnt/net/bat710` idem
Etape 9 : `/tmp_mnt/net/bat710/disc0` doit passer la main a NFS
Etape 10 : `/tmp_mnt/net/bat710/disc0/ftp` est bien un directory sur `bat710`
Etape 11 : la commande `ls` fait son boulot en récupérant les informations liées à ce directory.

### 5.5.4 AFS : Andrew File System

Très similaire à NFS mais les fichiers sont "cachés" localement. Il faut donc retrouver ou ce trouve la dernière version du fichier.

### 5.5.5 DFS : Distributed File System

Outil DCE s'inspirant de AFS.

### 5.5.6 XFS

Architecture complètement différente dans laquelle l'ensemble des serveurs coopèrent afin d'offrir un stockage sur l'ensemble des machines complètement transparent. C'est comme si l'on avait un gros disque sur l'ensemble du réseau. Evidemment les données sont dupliquées et migrent afin d'avoir de bonnes performances.

En fait c'est une architecture sans serveur (ou tout serveur...).

# 6 Exemples, 1 serveur, assynchrone, avec état

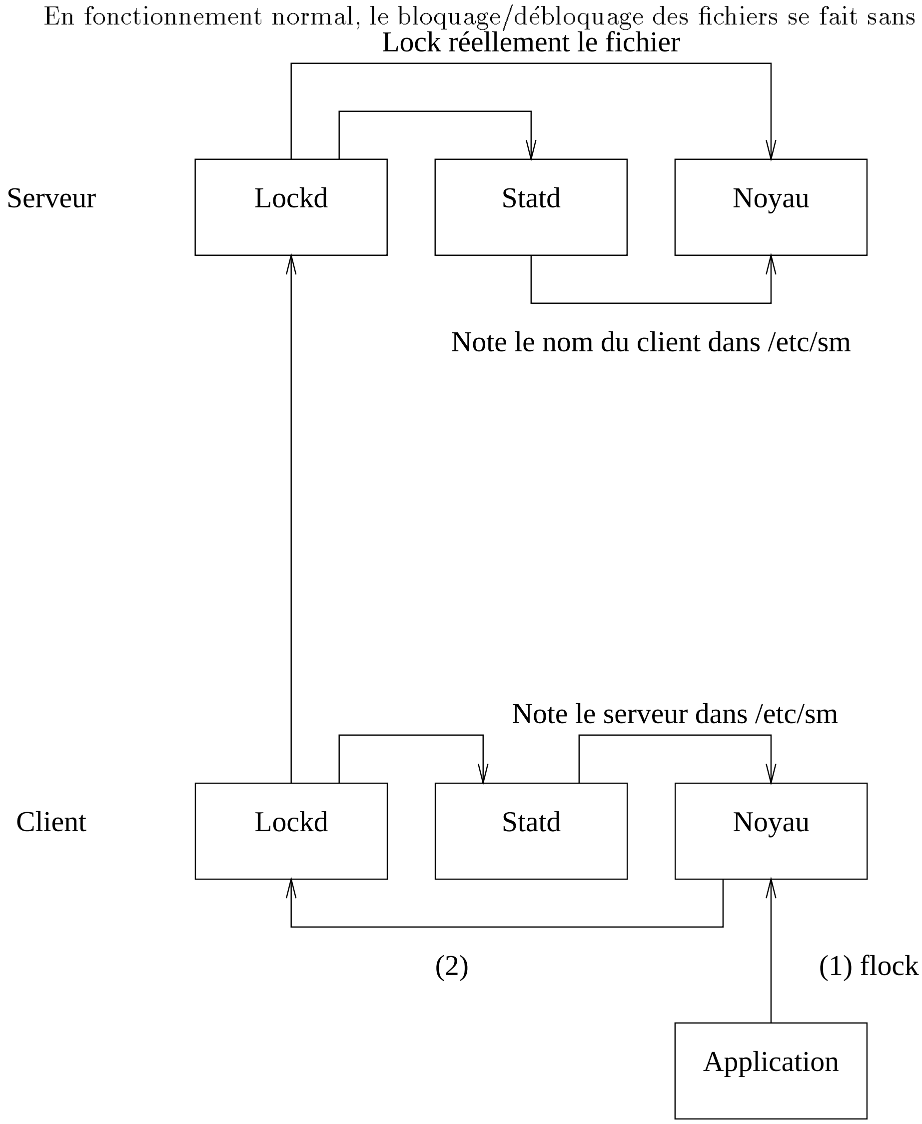
## 6.1 LOCKD: NFS file locker

Le système UNIX a une fonction (`flock`) permettant de blocker un fichier en tout ou partie.

Pour une utilisation transparente des fichiers à travers le réseau, il faut que les fonctions de bloquage fonctionnent, hors NFS est un protocol sans état et ces fonctions n'ont pas été incluses dans NFS.

Sur chaque machine cliente ou serveuse NFS, un démon nommé `rpc.lockd` est chargé de s'occuper de locker les fichiers. Il reçoit les requêtes de `lock` et les fait passer au bon serveur, si local, il `lock` lui-même. De plus, il indique au démon `statd` qu'il faut surveiller le serveur distant.

En fonctionnement normal, le bloquage/débloquage des fichiers se fait sans problème.




Mais du fait que le protocol est à état (un fichier reste bloqué ou pas) tout devient plus compliqué en cas de panne ou d'arrêt de machine.

- Le client s'arrête brutalement sans prévenir le serveur pour débloquer les fichiers.
  Lors de son reboot le client prévient tous les `lockd` de serveurs afin qu'il libèrent les bloquages.
  Ceci est possible car `statd` a gardé sur disque la liste des serveurs.
- Le serveur s'arrête.
  Lors de son reboot, il prévient tous les clients (connus grace à `statd`) qu'il a perdu tous les bloquages.
  En retour, les clients redemandent au serveur de rebloquer les fichiers. Ceci est possible car le `lockd` du client à la liste de tous les blockages.

La liste des blockages est en mémoire et non sur disque car si elle disparait de la mémoire, cela veut dire que les autres processus se sont aussi arrêté et donc que les blockages ne sont plus nécessaires.

Pour durcir le protocol, les `statd` notent dans le fichier `/etc/state` un numéro d'ordre incrémenté à chaque reboot.

# 7 Exemples, 1 serveur aussi client, synchrone

## 7.1 Proxy

*Proxy is a term that originated in the legal community to indicate an entity empowered to perform actions on behalf of another.*

Les problèmes à résoudre sont le Firewall, non-Internet host et le cache.

La solution dans les 3 cas est le Proxy.

Le client, au lieu de de s'adresser au serveur s'adresse au proxy (ceci peut être fait de manière transparente en cas de routage). Le proxy, lui, s'adresse au serveur et relais l'information en sens inverse.

Le proxy gère les droits et le cache.

L'implémentation est dépendente du logiciel utilisé, de nombreux proxy spécifiques existent. Mail (MX), FTP, Telnet, X11, HTTP. Ces proxy «intelligent» peuvent servir de cache.

Mais voila quelques types génériques (SOCKS)

### 7.1.1 Routeur/Filtreur

Le traffic entre les deux réseaux passe par une machine qui joue le role de routeur. Celle-ci peut examiner les paquets et les modifier si besoin : IP masquarading, Filtre d'accès, Redirection, ...

Ce genre de filtre nécessite souvent une modification du noyau du système.

### 7.1.2 Forward

L'idée est simple : un faux client se met en attente sur le Proxy, quand on accède au proxy il passe le message au serveur réel qui peut être d'un coté ou de l'autre du proxy.

Ce genre de proxy s'installe directement au niveaux du socket et monitor/log le traffic dans les 2 sens. On n'a pas besoin de modifier le noyeau du système.

Ceci est bon pour l'extérieur voulant accéder à l'intérieur. Un exemple est TCP WRAPPER

### 7.1.3 Encapsulation

L'encapsulation permet au gens de l'intérieur d'accéder aux machines extérieurs. Il leur faut des clients modifiés qui travaillent avec des sockets particuliers qui se connectent au proxy et lui indiquant l'adresse de connexion finale.

# 8 Cryptage/Authentification

Cryptography: crypter un texte que seul la personne connaissant la clef peut décrypter.
Le texte crypté doit être aléatoire (on le comprime avant par exemple)
Système à clef privé. La clef sert à crypté/décrypter, c'est la même.

## 8.1 Cryptage

Algorithme de César: Décalage des lettres de l'alphabet.

Crypter UNE fois, la clef a la longueur du texte à crypter et doit être *aléatoire*. Si elle est réutilisée, il est possible de la retrouver...

Crypteur par produit. On prend un bloc du texte a crypter et une opération avec la clef. On répète plusieur fois. La sortie est aléatoire. DES is the U.S. Government's Data Encryption Standard, a product cipher that operates on 64-bit blocks of data, using a 56-bit key.

Clef publique (1976): résoud le problème de l'oeuf et de la poule. On encrypte avec la clef public et l'on décrypte avec la clef privé. On peut signer un message en cryptant un message avec la clef privé, les gens ayant la clef public pourront décrypter la signature. Ceci passe par un fonction mathématique non-inversible. En fait comme ces algos sont long, ils servent à transférer une clef temporaire unique.

*RSA is a public-key cryptosystem defined by Rivest, Shamir, and Adleman. Here's a small example. See also [FTPDQ].*

- Plaintexts are positive integers up to $2^{512}$
- Keys are quadruples $(p,q,e,d)$
  - $p$ a 256-bit prime number
  - $q$ a 258-bit prime number
  - $d$ and $e$ large numbers with $(de-1)$ divisible by $(p-1)(q-1)$
- $E_K(P) = P^e \% pq$
- $D_K(C) = C^d \% pq$.
- Public : $(pq, e)$
- Private : $d$

*All quantities are readily computed from classic and modern number theoretic algorithms (Euclid's algorithm for computing the greatest common divisor yields an algorithm for the former, and historically newly explored computational approaches to finding large 'probable' primes, such as the Fermat test, provide the latter.)*

*Now $E_K$ is easily computed from the pair (pq,e)—but, as far as anyone knows, there is no easy way to compute D_K from the pair (pq,e). So whoever generates K can publish (pq,e). Anyone can send a secret message to him; he is the only one who can read the messages.* RSA est patenté aux états-unis.

La NSA (National Security Agencie) est le plus gros employeur de mathématiciens et le plus gros acheteur de machines puissantes.

PGP utilise RSA.

## 8.2 Signature

On veut signer un message. On fait du hashage non inversible de ce message. Il est impossible de modifier le message et d'obtenir le code de hashage.

Le hashage n'est pas local mais globale. Une somme n'est pas suffisante.

On obtient évidemment le code par un moyen fiable. Ou on l'encrypte avec une clef public.

Le plus utilisé actuellement: MD5 (RFC 1321)

## 8.3 Cassage

La clef n'est pas aléatoire ou elle peut être perdue.

Force brute pour casser le code, impossible car trop de clefs différentes. DES va bientot être cassable car il n'a que $10^{17}$ clefs.

Etre résistant au fait de trouver la clef si l'on a à la fois le texte original et crypté.

On peut capturer les images vidéo à distance facilement....

## 8.4 Authentification

L'authentification ce fait en passant un message crypté que l'on est la seul a pouvoir crypté

## 8.5 En France

Le chiffrement est pratiquement interdit en France par la loi 90-1170 du 29 décembre 1990 (avec décret d'application en décembre 1992).

1. Soit vous utilisez le chiffrement à seule fin d'authentification (chiffrement de mots de passe pour qu'ils ne circulent pas en clair sur le réseau, par exemple) : il faut faire une déclaration au SCSSI qui est ensuite vérifiée.

2. Soit vous l'utilisez pour la confidentialité. Il faut alors demander une autorisation au SCSSI.

J'y ajoute que les critères d'acceptation ou de refus ne sont pas publics. Il semble que l'autorisation ne soit donnée qu'aux institutions "sérieuses" (banques, ...) et à condition de ne pas utiliser de logiciels trop efficaces. En effet, il s'agit de conserver la possibilité pour la police de suivre les échanges. On notera que la plupart des pays comparables à la France autorisent le chiffrement,

## 8.6 Kerberos

Cela n'utilise pas le concept clef public, clef privée.

*The Kerberos model is based in part on Needham and Schroeder's trusted third-party authentication protocol [3] and on modifications suggested by Denning and Sacco [4]. The original design and implementation of Kerberos Versions 1 through 4 was the work of two former Project Athena staff members, Steve Miller of Digital Equipment Corporation and Clifford Neuman (now at the Information Sciences Institute of the University of Southern California), along*

*Kerberos provides a means of verifying the identities of principals, (e.g., a workstation user or a network server) on an open (unprotected) network. This is accomplished without relying on authentication by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets traveling along the network can be read, modified, and inserted at will. (Note, however, that many applications use Kerberos' functions only upon the initiation of a stream-based network connection, and assume the absence of any "hijackers" who might subvert such a connection. Such use implicitly trusts the host addresses involved.) Kerberos performs authentication under these conditions as a trusted third- party authentication service by using conventional cryptography, i.e., shared secret key. (shared secret key - Secret and private are often used interchangeably in the literature. In our usage, it takes two (or more) to share a secret, thus a shared DES key is a secret key. Something is only private when no one but its owner knows it. Thus, in public key cryptosystems, one has a public and a private key.)*

Mots-clefs :

- C : Le client, le demandeur de connexion.
- S : Le serveur
- K : Le serveur de clefs
- Clef session : clef privée permettant à C et S de communiquer de manière cryptée.
- Identité : Identité de C.
- Tiquet : (Identité de C + Clef session) crypté avec la clef de S
- Crédit : (Tiquet + Clef session) crypté avec la clef de C

Protocole :

- C demande à K un crédit pour communiquer avec S
- C reçoit le **Crédit** venant de K et le décrypte.
- C envoit le **Tiquet** à S
- S reçoit le **Tiquet**, le décrypte
- C et S peuvent donc communiquer avec la clef de session.

La clef de session n'a jamais circulée non-codé.

La clef de session permet alors de crypter les messages dans les 2 sens.

Pour évité la réutilisation de vieux ticket, la date est codée.

Les différents AS communiquent entre eux par la même méthode pour faire des tickets entre n'importe quelle paire de machines.

# 9 Exemples, réseau de serveurs, synchrone

## 9.1 DNS: Domain Name Service

C'est le démon gérant les noms de domaines. Ce sont les noms logiques (utilisateurs) de machine par exemple : `miage.univ-lyon1.fr`

La gestion centralisée des informations a été abandonnée vers 1987. La base était trop grosse.

La dernière partie du nom est appelée *top-domain*.

`named` est en fait la plus grande base de données répartie du monde, toute machine faisant fonctionner `named` participe à la gestion de cette base de donnée.

Cette base de données permet à partir du nom ou de l'adresse IP d'une machine d'obtenir des informations sur la machine.

### 9.1.1 Utilisation de `named`

Chaque fois que vous faite un `rlogin miage` il faut retrouver l'adresse IP de la machine dont le nom est `miage.univ-lyon1.fr` avant de pouvoir communiquer avec elle. Pour la trouver, on s'adresse au processus `named` en utilisant les `socket`. Voici un extrait du fichier `/etc/services`:

```
domain          53/tcp  nameserver      # Domain Name Service
domain          53/udp  nameserver      #
```

Pour avoir d'autres informations, voici un exemple d'utilisation de la commande `nslookup`

Attention le nom logique n'a aucun lien avec une position géographique, une machine peut avoir plusieurs nom logiques, un domaine peut être répartie sur l'ensemble de la planète.

```
Default Name Server:  ligiahp3.univ-lyon1.fr
Address:  134.214.88.60

======================================================> miage
Name Server:  ligiahp3.univ-lyon1.fr
Address:  134.214.88.60

Name:    miage.univ-lyon1.fr
Address:  134.214.88.23

======================================================> set type=any
======================================================> miage
Name Server:  ligiahp3.univ-lyon1.fr
Address:  134.214.88.60

miage.univ-lyon1.fr     preference = 5, mail exchanger = bat710.univ-lyon1.fr
miage.univ-lyon1.fr     preference = 10, mail exchanger = cismsun.univ-lyon1.fr
miage.univ-lyon1.fr     internet address = 134.214.88.23
bat710.univ-lyon1.fr    internet address = 134.214.88.10
cismsun.univ-lyon1.fr   internet address = 134.214.100.6
======================================================> ligia
Name Server:  ligiahp3.univ-lyon1.fr
Address:  134.214.88.60

ligia.univ-lyon1.fr     canonical name = ligiahp.univ-lyon1.fr
======================================================> set type=ptr
======================================================> 134.214.88.10
Name Server:  ligiahp3.univ-lyon1.fr
Address:  134.214.88.60

10.88.214.134.in-addr.arpa      name = bat710.univ-lyon1.fr
```

Avec cette commande on peut aussi lister toutes les machines d'un domaine.

### 9.1.2 Buts

Ne jamais tomber en panne, être à jours, évolutif, n'avoir aucune limites.

Jouer le role de cache afin de ne poser sans cesse les même questions, ne pas s'adresser sans arrêt au chef des chef.

### 9.1.3 Données

Chaque `named` dispose:

- d'informations sur les informations dont il dispose. Généralement dans le fichier `/etc/named.boot` Il indique aussi ou stocker les informations qu'il connait de seconde main.

```
cache                        .                /etc/named.ca    # topdomain cache
secondary        univ-lyon1.fr 134.214.100.6 /etc/named.hosts # stock 134.214
secondary 214.134.IN-ADDR.ARPA 134.214.100.6 /etc/named.rev   # stock 134.214
primary   0.0.127.IN-ADDR.ARPA               /etc/named.local
forwarders 134.214.100.6 134.214.100.245 134.214.84.25 134.158.69.104
```

  Dans cet exemple la machine est un serveur secondaire pour le domaine `univ-lyon1.fr`
- `/etc/named.ca` indique les adresses des machines qui connaissent les *top-domain*

```
.                         3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.       3600000      A     198.41.0.4
.                         3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.       3600000      A     128.9.0.107
```

- `/etc/named.hosts`

```
$ORIGIN fr.
univ-lyon1      IN      SOA     cismsun.univ-lyon1.fr. postmaster.univ-lyon1.fr. (
                1995112103 14400 14400 12096000 604800 )
                IN      NS      cismsun.univ-lyon1.fr.
                IN      NS      cismhp.univ-lyon1.fr.
                IN      NS      ccpnvx.in2p3.fr.
                IN      NS      ccpntc3.in2p3.fr.
                IN      MX      10 cismsun.univ-lyon1.fr.
$ORIGIN univ-lyon1.fr.
jacquemo        IN      A       134.214.103.12
iutainfopc10    IN      A       134.214.80.190
iutainfopc63    IN      A       134.214.80.243
cismhp          IN      MX      0 cismhp.univ-lyon1.fr.
                IN      MX      10 cismsun.univ-lyon1.fr.
                IN      A       134.214.100.245
lpnsun1         IN      CNAME   neurosens.univ-lyon1.fr.
ura1195-1       IN      A       193.48.219.10
radioth11       IN      A       134.214.8.111
```

- `/etc/named.rev` est le fichier inverse du précédent.
- `/etc/named.local` est le fichier qui contient les informations réellement connues par le `named` local.

```
$origin 0.0.127.in-addr.arpa.
@ 604800 IN SOA LOCALHOST. postmaster.localhost. (
        1 ; serial
        7200 ; refresh 2 heures
        7200 ; retry 2 heures
        12096000 ; expire 20 semaines
        86400 ) ; minimum
;
        604800  IN NS LOCALHOST.
1               IN      PTR     LOCALHOST.
LOCALHOST.      IN      A       127.0.0.1
```

  Cet exemple est stupide car il ne contient pas d'information. En effet seule une seule machine du domaine contient la véritable base de données, les autre contiennent des copies.
- Les *forwarders* sont les machines à qui poser la question quand on n'arrive pas à trouver la solution seul.

### 9.1.4 Algorithmes

Plusieurs sont possibles, en voici un:

1. Regarde si la donnée est dans la cache
2. Regarde si la donnée est connue localement «primairement».
3. Regarde si la donnée est connue localement «secondairement».

4. Regarde si l'on sait à qui s'adresser pour demander l'information. Par exemple si l'on cherche la machine `toto.insa-lyon.fr` et que l'on connait le NS du domaine `insa-lyon.fr`
5. Pose la question aux *forwarders*

Les pas 1,2,3 éliminent les accès réseaux.

Le pas 4 a l'avantage de ne pas surcharger le `forwarder` en posant la question directement au bon NS. Il a comme inconvénient de plus charger le bon NS et de ne pas remplir le cache du `forwarder`, générant donc plus de traffic entre les 2 réseaux.

L'inconvénients de la surcharge du pas 4 peut être résolu en donnant plusieurs NS pour un même `domain`, on choisi alors aléatoirement celui auquel on s'adresse. (Technique aussi appliquée en donnant plusieurs adresses IP a un seul nom de machine, on choisi aléatoirement l'adresse IP, ceci équilibre la charge)

Les *forwarders* sont obligatoires si l'on ne peut communiquer directement avec le bon NS (*FireWall*).

En remplissant les fichiers de configuration on peut diriger l'algorithme et ainsi donc répartir la charge.

## 9.2 Sendmail

C'est le démon permettant l'acheminement du courrier électronique. Ce démon est complexe, multiforme et à tendance à rendre les humains perplexes.

Quelques données de base :

- `sendmail` peut utiliser plusieurs méthodes de transport des messages.
- Il ne doit pas perdre de messages.
- Il doit être économe sur les liens de communications.
- Il doit fonctionner sur n'importe quel système d'exploitation.

Tout ceci est très difficile à mettre en oeuvre, le programme a donc été simplifié et le langage de programmation le plus illisible possible a été créé afin de le configurer. Le fichier de configuration est `/usr/lib/sendmail.cf`

Le démon reçoit les messages de la couche transport, si celle-ci est TCPIP, il les reçoit sur le port 25.

Par analyse du message, il en déduis à qui il est adressé, si c'est local alors il stock le mail dans une boîte à lettre, sinon il analyse l'adresse et redirige le mail vers une machine plus proche de la destination.

Si le mail n'est pas délivrable, `sendmail` va essayé régulièrement de le redélivrer. Si c'est impossible, au bout de quelques jours, le mail revient à l'expéditeur (c'est généralement possible).

Lorsque le mail est délivré par TCPIP, il est directement envoyé à la machine réceptrice ou à son MX. Le MX de toutes les machines du bâtiment 710 est `bat710` c'est lui qui contient les boîtes aux lettres. Si le MX n'est pas disponible, on essaye d'envoyer le message à un MX de plus faible priorité (`cismhp` par exemple).

Si les messages doivent passer par des liaisons téléphoniques, il peut être interressant d'attendre d'en avoir plusieurs avant de composer le numéro....

Si les messages passent par UUCP (Unix to Unix CoPy) qui est le prédécesseur d'Internet. Avec UUCP la notion de routage n'existe pas, l'adresse contient en elle-même la route: `exco@hostA!hostB!hostC` Pour communiquer, il fallait connaitre l'une des machines du chemin.... Ceci n'existe plus dans les pays développés ou l'on a de quoi se payer des liaisons fixes. Et en plus c'est pas vrai, l'exemple pris au hasard date de 1993!

En outre du fait de pouvoir délivrer des messages dans les boîtes aux lettres, il peut faire exécuter des programmes automatiquement lors de la réception de messages.

### 9.2.1 Le format des messages

RFC822 (1982, 2900 lignes) décrit le contenu des messages possibles. Son extension (MIME) permet de faire du multimedia.

Un champ est un type suivit de ':' puis la contenation de toute les lignes suivantes qui commencent pas par un caractère.

```
Return-Path:
Received:
Date:
From:
Subject:
Sender:
To:  "Joe & J. Harvey" <ddd @Org> (commentaire), JJV (commentaire2) @ BBN
Comments:

texte du message
```

Les caractères spéciaux sont quotés par '"' ou entre guillemets.

Majuscules/minuscules sont interchangeables.

Exemple d'entête de message :

```
From daemon Mon Feb  1 20:30 GMT 1993
```

```
Received: from cism.univ-lyon1.fr
          by ligiahp.univ-lyon1.fr (15.11/4.04)
          id AA17324;
          Mon, 1 Feb 93 20:30:40 lyo
Received: from uu3.psi.com
          by cism.univ-lyon1.fr (PMDF #3026 )
          id <01GU7Y20EBTC8WYCMF@cism.univ-lyon1.fr>;
          Mon, 1 Feb 1993 20:30:53 +0200
Received: from crynwr.UUCP
          by uu3.psi.com (5.65b/4.0.071791-PSI/PSINet)
          via UUCP;
          id AA22932
          Mon, 1 Feb 93 13:51:21 -0500
Received: by crynwr.com (UUPC/extended 1.11q);
          Mon, 01 Feb 1993 13:44:21 EST
Date: 01 Feb 1993 13:44:20 -0500 (EST)
From: Russell Nelson <nelson@crynwr.com>
Subject: fsb addition
To: mass-fsb-acknowledgement@crynwr.com
Message-Id: <2b6d6f86.crynwr@crynwr.com>
Organization: Crynwr Software
X-Envelope-To: exco@ligiahp.univ-lyon1.fr
Content-Transfer-Encoding: 7BIT
Status: RO
```

## 9.3 Hyper-G

Un remplacement de WWW pour résoudre quelques-uns de ses défauts.

- Liens pendant
- Recherche difficile
- Clients intelligent et lourd
- Mauvaises performances (pas de cache)
- En read-only
- Les hyper-liens sont dans le texte (lisibilité)
- Pas possible d'ajouter des liens a un document que l'on ne peut modifier
- Pas possible de suivre les liens dans l'autre sens
- Pas possible d'avoir des paquets de documents sans liens

Hyper-G résoud ces défauts en:

- Ayant la notion d'utilisateur et de groupe. on peut donc associer des droits à ceux-ci.
- Ne stockant pas les informations chez l'utilisateur mais chez le serveur:
  - L'utilisateur ne peut pas faire d'action sans mise à jour des bases de données du serveur. On est donc sur de toujours avoir de bons liens.
  - L'utilisateur peut modifier des informations sur un serveur distant
- Ne stockant pas les liens dans le fichier d'information
  - L'utilisateur peut faire des ajouts de liens sans modifier le fichier. Et donc sans en être propriétaire.
  - L'ajout de liens peut être simplement fait sur du texte quel que soit son type: des images, du postscript, des animations, ... Il suffit d'avoir une notion de "zone".
  - On peut faire de liens dans un fichier sur CDROM par exemple
- N'ayant des clients se raccordant à un unique client Hyper-G qui est sur le réseau local.
  - Le serveur sert de cache
  - Le client ne connait qu'un seul protocol de communication. C'est le serveur qui va évoluer.
  - Le client n'ouvre qu'une seule liaison et ne perd donc pas de temps à initialiser et stopper des liaisons avec des serveurs éparpillés.
- Ayant des structures de documents plus évoluée.

  Notemment les *collections* qui sont des ensembles de documents ou de pointeurs sur documents sur lesquels le SERVEUR peut faire une recherche. Ou déléguer sa recherche en parallèle sur les différents serveurs contenant l'information.

  Et des champs de mot-clef et de langages associés au document, sans parler évidemment des droits d'accès.

L'architecture est donc des paquets de clients sur des réseaux locaux (avec un serveur local) et un réseau de serveur coopératif.

Le lecteur Hyper-G est (sur les 9000/700) «harmony»

# 10 Exemples, 1 serveur, asynchrone

## 10.1 X11

Toute les documentations techniques sont dans `/vol/X11R6.src/xc/doc/`

Le protocol X11 est simple, comme tous les autres il utilise les sockets (un bitstream comme un autre). Vous pouvez donc faire `telnet nom_machine 6000` pour vous connecter sur le premier serveur X11 de la machine.

Le client essaye d'abord d'ouvrir un socket local avant d'ouvrir un socket réseau.

Pour des raisons d'afficacité le protocol est en binaire en non en ASCII lisible par un être humain. Il est donc difficile de faire quoique ce soit avec la commande `telnet` à moins bien sûr de rejouer une session.... En faisant `telnet nom 6000 <a_envoyer >reçu`

La communication est asynchrone. On peut recevoir des réponses à n'importe quel moment, on n'est pas obligé d'attendre la réponse à une question, on peut en poser plusieurs de suite ou demander à exécuter des fonctions.

Une communication asynchrone permet un gain de vitesse pouvant être énorme. Imaginez seulement le client et le serveur séparés par une liaison satellite! On pose 10 questions synchrones: 10*(1/4+1/4) soit 5 secondes, en asynchrone: 1/2 seconde! Je néglige évidemment le débit.

*Le serveur X11 est lui même un client du serveur de fontes...*

Le serveur X11 assure la communication entre ses clients. Le plus couramment, pour le copié/collé.

*Request Format*:

- 1 octet: major opcode,
- 2 octets: nombre de 4 octets y compris celui-ci,
- 1 octet: premier octet du message

Le serveur numérote toutes les requêtes reçues,

*Reply Format*: au moins 32 octets comprenant:

- 4 octets: nombre de 4 octets
- 2 octets: numéro de la requête déclenchant la réponse (partie basse)

*Error Format*: 32 octets de long:

- 1 octet: numéro erreur
- 1 octet: major opcode
- 1 octet: minor opcode
- 2 octets: numéro de la requête
- the failing resource ID is also returned: Colormap, Cursor, Drawable, Font, GContext, IDChoice, Pixmap, Window, Atom, Value.

*Event Format*: 32 octets de long

- 1 octet: code événement
- 2 octets: numéro de la requête

## 10.2 Introduction à X

X ou X11 ou X window system. Le 11 vient du fait que l'on est en version 11. La release est la sixième

X est un système de multifenêtrage réseau, il a été par le MIT, le projet a été commencé en 1985 La diffusion à grande échelle en 1988. X est diffusé gratuitement avec les sources, c'est maintenant un standard de fait.

Mot-clefs concernant le serveur :

- Architecture client/serveur, le serveur gère la ressource écran(s) et périphériques d'entrées.
- Le serveur peut communiquer avec plusieurs clients sur des machines différentes.
- L'application fonctionnera de la même manière quelque soit l'écran, qu'il soit local ou non.
- Le serveur permet toutes les manipulations bas niveaux sur l'écran. Il travaille au niveau pixel.
- Le serveur peut fonctionner sur tous les types d'écran matriciels.
- Le serveur gère une *hiérarchie de fenêtres*.
- Le *window manager* est un client comme un autre, il est indépendant du serveur.
- Le serveur gère des ressources, ces ressources sont partageables entre les clients :
  - Les fenêtres (Window).
  - Les tables de couleurs (Colormap).
  - Les contextes graphiques (GC).
  - Les matrices de bit ou de pixel (Bitmap,Pixmap).
  - Les fontes (Font).
  - Les curseurs (Cursor).
  - Des tampons de données (Cutsel).
  - Une base de données de configuration d'applications (xrdb).

  - Les symboles clavier (KeySym).
  - Type d'affichage (Visual).
- Le serveur offre des mécanismes de base pour échanger des données entre des clients.
- Le serveur peut contenir des extensions :
  - PEX pour supporter PHIGS.
  - VEX pour l'affichage d'une vidéo extérieur.
  - XIE pour le traitement d'image.
  - ...
- Le serveur peut directement être accédé via la bibliothèque `Xlib.a` celle-ci offre les fonctions élémentaires permettant de tous faire, c'est l'*assembleur X*.
- Les intrinsics (`libXt.a`) sont des fonctions permettant une approche objet de la conception d'une application.
- On appel *Widget* les objets que permet de créer `libXt.a`.
- MOTIF est un gestionnaire de fenêtres et un ensemble de *Widget* MOTIF et n'est pas gratuit.
- Athena Widget est une bibliothèque de *Widget* très employée elle est puissante et gratuite.

### 10.2.1 A l'intérieur de X

Quand on ouvre une fenêtre on peut utiliser les visuals suivants : Static color, Pseudo color, Static gray, Gray scale, True color, direct color.

Les événements : KeyPress, KeyRelease, ButtonPress, ButtonRelease, MotionNotify, EnterNotify, LeaveNotify, FocusIn, FocusOut, Expose, UnmapNotify, MapNotify, ConfigureNotify, ResizeRequest, Visibility, ... Si l'événement provient d'un périphérique il est envoyé à la plus petite fenêtre contenant le curseur, si celle-ci n'en veut pas, le serveur essaye avec le père et ceci jusqu'a la racine. Ceci est faux en cas de "Focus window" et de Grab sur un périphérique.

Les attributs graphiques (GC : Graphic Context) sont très nombreux : Fonction, plane_mask, foreground, background, line_width, line_style, cap_style, fill_style, tile, stipple, subwindow_mode, font, ...

Les attributs de fenêtres : background_pixmap, background_pixel, border_pixmap, border_pixel, win_gravity, backing_store, save_under, event_mask, do_not_propagate_mask, colormap, cursor, ...

Les fonctions d'affichages : Manipulations de rectangles de pixel, affichage de points, lignes et arcs de cercles ainsi que de caractères.

Communiquer avec le window manager : nom de la fenêtre, taille, icone, colormap.

Définir des `Atom` les Atoms sont l'expression sous forme de chiffre d'une chaîne de caracteres qui représente un nom. Le serveur contient la correspondance entre le numéro de l'atome et la chaîne de caractères. De nombreux atomes sont prédéfinis.

Définir des `Properties`, chaque property est une donnée typée associée à une fenêtre par exemple : la propriété WM_NAME est de type STRING elle permet au window manager de trouver le titre de la fenêtre pour l'afficher.

Une sélection est un property qui n'appartient qu'a une fenêtre, cette property appartient à la fenêtre qui détient la selection. L'ancien possesseur est prévenu par un événement si on lui arrache sa sélection. Exemple : le coupé coller de texte.

Décodage clavier, chaque touche du clavier possède un numéro appellé `KeyCode` ce numéro dépend du clavier. A chaque Keycode correspond une liste de `Keysym` qui indiquent le code à retourner en fonction du `modifier` (shift, ctrl, ...). La liste de Keysim associée à chaque Keycode peut être modifié.

Il est possible d'envoyer des événements directement à une client ou bien lui voler des événements, ceci est notamment fait par le window manager lors de l'ouverture de la fenêtre ou bien lors du changement de sa taille.

Des fonctions utilitaires travaillant sur des régions, elle permettent de faire certain calcul localement pour éviter des accès au serveur.

331 fonctions, 114 structures et 349 constantes définie par `define`. Accessibles par `Xlib.a`

Une première couche au-dessus de `Xlib.a` permet de créer de `Widget` se sont les intrinsics `libXt.a`. Cette couche permet de créer des applications plus simplement, elles sont d'autre part dans le même "moule". Cette couche n'est pratiquement jamais utilisée par le programmeur, elle programmée sous forme d'objets de classe et de méthode. Elle est très maléable et comporte de nombreuse possibilités.

La couche au-dessus `libXaw.a` comporte des instances de Widget. Elle est très facilement utilisable, on définit une hiérarchie de Widget avec leur propriétés. On appel ensuite la boucle principale, les intrinsics se chargent de tout. Les Widgets peuvents être des Label, Menu, Boutton, .... La programmation est d'un très haut niveau. L'utilisateur peut facilement configurer son application. Pour comprendre cela : lancer `editres` sur une application utilisant les Athena Widget.

### 10.2.2 Athena Widget

**Utilisation des `widgets`** :

Un `widget` est un zone rectangulaire de l'écran possédant un certain comportement.

Chaque `widget` est forcément inclus dans un autre rectangle, il possède donc un père.

Un `widget` porte un nom court et un nom long formé par la concaténation des noms de ces ancêtres et du sien. Le spéparateur est le ".".

Un `widget` *composite* peut contenir d'autres `widget`. Il dispose ses fils selon un algorithme qui lui est propre.

Une ressource est définie par un nom de `widget` suivit d'un nom de ressource et de ":" et la valeur de la ressource.

Un widget peut être complètement défini par ces ressources.

```
MonApplication.menu.quit.label: Quit
MonApplication.menu.quit.foreground: Red
MonApplication.menu.quit.font: -*-courier-*-r-*-*-12-*-*-*-*-*-iso8859-*
```

Ceci n'est qu'une petite partie des ressources.

On peut spécifier les ressources à l'aide de patterns : l'"*" remplace une chaîne `widget` et le "?" un seul. Précision et ordre des ressources sont importants.

L'application "dialogue" avec les `widget` en utlisant les fonctions "GetValues" et "SetValues" qui permettent de modifier durant l'exécution les ressources du `widget`. Il est mis à jour automatiquement sur l'écran.

Un `callback` est un fonction C qui est appelée chaque fois que le `widget` est dans un état "particulier" qui n'est pas lié à un événement précis. Par exemple pour le bouton de commande.

Une action est une fonction C dont on a permis l'appel direct par les `widget`

Un `widget` est capable d'executer certaine actions en fonction d'événements qui lui arrivent : (<Key>, <BtnDown>, <Map>, <EnterWindow>, <Leave>, <Btn1Down>, <Btn1Up>, <Btn3Motion>, ... ) ou d'une suite d'événements :

```
*Text.translations : #override\
                 Shift<Key>Find: search(backward)\n\
                 <Key>Find:      search(forward)\n\
                 <Key>Delete:    delete-next-character()\n\
                 <Btn3Down>,<Leave>,<Btn3Up> :      select-all()\n\
                 <Key>Next:      next-page()\n\
                 <Key>Prior:     previous-page()
```

Une application définira la hiérarchie des `widget`. Elle associera des resources aux différents `widgets`.

Exemples de fichiers de ressources dans :
/vol/X11R6/lib/X11/app-defaults

Modification des ressources par `xrdb` ou `editres`. Elles sont recherchées dans : le terminal, app-defaults, le programme.

**Création de nouveaux `widgets`** :

Il sont organisé sous forme d'une hiérarchie avec héritage des méthodes et des attributs. Par exemple Object, Rect, Core, Simple, Label, Command, MenuButton.

On crée un nouveau `widget` en spécialisant son comportement. Il récupère tout ce que ses ancêtres savent faire.

Le nouveau `widget` peut s'ajouter de nouvelles ressources qui seront connue par sa descendance.

Il ajoute de nouveaux `callback`.

Visualisation de la hiérarchie par `viewres`.

`widgets` **existant** :
SIMPLE : Command, Toggle, Label, AsciiTexte, List, Panner, ScrollBar, ...
COMPOSITE : Box, Dialog, Form, Paned, Porthole, ...

### 10.2.3 Exemple programme XAW

Dans `/home/exco/LOGICIELS/XAW`
Le `Imakefile`

```
LOCAL_LIBRARIES = XawClientLibs
SRCS1 = main.c toto.c
OBJS1 = main.o toto.o
PROGRAMS = main
CC=gcc
CCOPTIONS= -g -Wall -pedantic -pedantic-errors

ComplexProgramTarget_1(main,$(LOCAL_LIBRARIES),-lm)

main.o:toto.h
toto.o:toto.h
```

Puis on lance `xmkmf` pour créer le `Makefile`, puis on fait `make`
Le fichier source simplifié contient :

```
/* Include files required for all Toolkit programs */
#include <X11/Intrinsic.h>/* Intrinsics Definitions */
#include <X11/StringDefs.h>/* Standard Name-String definitions */
/* Public include file for widgets we actually use in this file. */
#include <X11/Xaw/Command.h>/* Un bouton de commande */
#include <X11/Xaw/AsciiText.h>/* Une zone de texte editable */
...
/************************************************************************/
/* Le programme principal */
/************************************************************************/
int main(argc, argv)
int argc;
char **argv;
{
/************************************************************************/
/* Ce tableau permet de faire le lien entre une fonction C et une  */
/* chaine de caractere (surtout l'inverse)                          */
/* ou par la commande editres */
/************************************************************************/
static XtActionsRec mesactions[] = {
{"Quit"          , Quit},/* Sortie de l'application */
{"Popup"         , Popup},/* Affichage d'une fenetre popup */
};
/************************************************************************/
/* Les ressources suivantes permettent de configurer l'application */
/* Elle peuvent etre modifiees avec "xrdb" ou par un "app-default" */
/* ou par la commande "editres" */
/* On peut en fait completement enlever ce tableau du programme */
/* principale pour le mettre dans un fichier ASCII */
/************************************************************************/
static String fallback_resources[] = {
/* Genericite sur le positionnement dans un widget Form */
"*tout.?.width:500",/* Largeur widget dans tout */
"*translations : #augment\\n\
 Ctrl<Key>c:Quit()",/* pour tous, le sortie ^C */
/* Genericite sur les fenetres infos */
/* La genericite est obtenue en donnant le meme nom
   a des widgets etant a des positions differentes dans la hierarchie.
   On les differencie par leurs parent
*/
"*info*but.label : Ferme fenetre",/* Le close fenetre */
"*info*txt.translations : #override <Key>:Popdown()", /* Change l'action */
```

```
/* La fenetre "nb" */
"*menu.nbwin.dialo.label: Nombre d'iterations entre les affichages",
"*menu.nbwin.dialo.value: 100",/* Valeur par defaut */
"*menu.nbwin.dialo.icon: woman", /* Nom de l'icone */
/* le return fait un apply */
"*menu.nbwin.dialo*translations: #override \\n <Key>Return:Applique()",
"*menu.nbwin.title : Valeur de NB",
NULL /* Important car il indique la fin de la liste de String */
};
/*********************************************************************/
/* Les options de la ligne de commande pouvant directement modifier  */
/* les valeurs des ressources precedentes                            */
/*********************************************************************/
static XrmOptionDescRec options[] = {
{"-gx","*graphic*width",   XrmoptionSepArg, NULL},
{"-gy","*graphic*height",  XrmoptionSepArg, NULL},
{"-fq","*quit*foreground", XrmoptionSepArg, "Red"}
} ;
/***********************/
/* Variables locales */
/***********************/
XtAppContext app_context; /* L'application */
Widget
topLevel,/* L'application */
tout,/* Toute la fenetre */
menu,/* La zone menu  "form" */
vdt ;
/*********************/
/* Debut du code reel */
/*********************/
topLevel = XtAppInitialize(
&app_context,        /* Application context */
"XSystemeSolaire",           /* Application class */
options,XtNumber(options),      /* command line option list */
&argc, argv,         /* command line args */
fallback_resources,             /* for missing app-defaults file */
NULL,0);             /* terminate varargs list */

/* Nom, Classe, Pere, Arguments de creation , nombre d'arguments */
tout    =XtVaCreateManagedWidget("tout"   ,formWidgetClass   ,topLevel,NULL);

/* Dans tout */
menu    =XtVaCreateManagedWidget("menu"   ,formWidgetClass     ,tout    ,NULL);
graphic =XtVaCreateManagedWidget("graphic",graphicWidgetClass  ,tout    ,NULL);
dt      =XtVaCreateManagedWidget("dt"     ,scrollbarWidgetClass,tout    ,NULL);
vdt     =XtVaCreateManagedWidget("vdt"    ,labelWidgetClass    ,tout    ,NULL);

/* Dans le menu */
fichier =XtVaCreateManagedWidget("fichier",menuButtonWidgetClass,menu,NULL);
parames =XtVaCreateManagedWidget("parames",menuButtonWidgetClass,menu,NULL);
help    =XtVaCreateManagedWidget("help"   ,commandWidgetClass   ,menu,NULL);

/* On donne la correspondance entre les chaines de caracteres et
   les fonctions C a appeler
 */
XtAppAddActions(app_context, mesactions, XtNumber(mesactions) );

/* Ajoute un fonction a executer en arriere plan */
donnees.graphic  = graphic ; /* Pour l'affichage */
donnees.contexte = app_context ; /* Pour reenclencher le WorkProc */
donnees.vdt      = vdt ; /* Pour changer le label ascenseur */
donnees.gc       = 0   ; /* Couleur */
XtAppAddWorkProc(app_context,arriere_plan,&donnees) ;
```

```
/* Les callback */
/* XtNcallback est une action a faire sur un objet,
   Sur un bouton et menubutton elle est declenche par l'appui
   Sur un element de menu, par le relachement

   XtNgraphicExpose est appele a chaque exposition
   XtNjumpProc est appele a chaque modification de l'ascenseur

   On peut mettre plusieurs callback sur le meme objet
*/
XtAddCallback(quit      ,XtNcallback     , Quit      , NULL);
XtAddCallback(apropos   ,XtNcallback     , Popup     , NULL);
XtAddCallback(aproposbut,XtNcallback     , Popdown   , NULL);

/* La fonction suivante fait mettre a jour la valeur de "nb"
   en fonction de la valeur courante du widget
   Je trouve ca plus propre que d'appeler directement le callback
*/
XtCallCallbacks(nbapply,XtNcallback,&donnees) ;

/* Create windows for widgets and map them.  */
XtRealizeWidget(topLevel);

/* Loop for events. */
XtAppMainLoop(app_context);

return(0) ; /* Pour eviter un Warning du compilateur, on ne passes pas ici */
}
```

### 10.2.4 Motif

Une bibliothèque de Widget bien conçue (mais mal programmée), un compilateur d'UIL et un gestionnaire de fenêtres.

Mieux qu'Athéna:

- Input focus sur un widget
- Drag and Drop
- Plus de Widget évolués (File Chooser)
- Un lange de plus haut niveau pour définir l'application

Moins bien qu'Athéna:

- Pas de configuration dynamique
- Payant
- Implémentation buggée

Exemple d'IUL pour une application affichant un texte et un bouton. Il est évidemment possible d'avoir des modules externes des variables typées, ...

```
! (c) Copyright 1989, 1990, 1991, 1992 OPEN SOFTWARE FOUNDATION, INC.
! ALL RIGHTS RESERVED
!
! Motif Release 1.2
!
!   $RCSfile: hellomotif.uil,v $ $Revision: 1.1 $ $Date: 1992/05/08 18:13:28 $
!*  (c) Copyright 1989, DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
!*******************************************************************************
module helloworld
    version = 'v1.0'
    names = case_sensitive

procedure
    helloworld_button_activate();

object
    helloworld_main : XmBulletinBoard {
        controls {
```

```
            XmLabel             helloworld_label;
            XmPushButton        helloworld_button;
        };
    };

object
    helloworld_button : XmPushButton {
        arguments {
            XmNx = 15;
            XmNy = 60;
            XmNlabelString = compound_string('Hello',separate=true) & 'World!';
        };
        callbacks {
            XmNactivateCallback = procedure helloworld_button_activate();
        };
    };

object
    helloworld_label : XmLabel {
        arguments {
            XmNlabelString = compound_string('Press button once',separate=true) &
                             compound_string('to change label;',separate=true) &
                             'twice to exit.';
        };
    };

end module;
```

# 11 Exemples, multiples serveurs accédés par un client

## 11.1 Netscape

Pour assurer une plus grande rapidité, ce produit ne se connecte pas successivement aux différents serveurs possédant des informations l'interessant mais en parrallèle. Cette méthode donne une meilleur sensation d'interactivité à l'utilisateur de NETSCAPE au détriment évidemment des autres utilisateurs.

La réalisation d'un tel logiciel est simple, sa structure en est la suivante :

```
Faire les requêtes aux différents serveurs

POUR TOUJOURS

    Attente bloquante d'un évenement X11 ou d'un des serveurs

        SI evénement X11
               ALORS
               SI clique: change de document
               SI affiche/scroll: réaffiche le document
               FIN SI

SI arrivée de données d'un des serveurs
               ALORS
               intégrer les données dans le modèle.
               SI pas de nouvelles données: réaffiche le document
               FIN SI

FIN POUR TOUJOURS
```

Ce type de client n'est possible qu'avec une gestion événementielle, sinon la programmation en est trop complexe.

# 12 Exemples, Répartition de charge entre serveurs

## 12.1 NIS, anciennement YP Yellow Page

Dès lors que l'on veux gérer des utilisateurs sur plusieurs machines et que l'on veut que les utilisateurs puissent se connecter indiféremment sur les différentes machines, il faut avoir des bases données pour définir les utilisateurs.

Une possibilité serait de dupliquer ces bases de données sur toutes les machines. Mais cela pose quelques petit problèmes :

- Il faut dupliquer l'informations sur toutes les machines et cela prend du temps. 10000 utilisateur, `passwd` fait 1Mo, s'il y a 1000 machines, cela fait 1Go à transférer sur le reseau. (multicast?)
- Certaines informations sont mise à jour souvent, ajout/destruction utilisateur ou changement de mot de passe. A chaque fois il faut retransférer la base de donnée

Le solution NIS est simple:

- Un serveur originel contenant les bases de données.
- Des serveurs esclaves contenant des copies des bases de données.
- Des clients qui s'adresse au premier serveur qui répond en lui dmeandant la partie de la base de donnée qui l'interesse.

A chaque modification de la base, elle est transférée sur chacun des serveurs secondaires.

La répartition de charge est un effet de bord ainsi que la reprise en cas de panne.

Une démon suplémentaire est `yppasswdd` qui recois les demandes de changements de mots de passe.

Sur le réseau, seul les mots de passe encryptés circulent.

### 12.1.1 Configuration UNIX

Sur le serveur principal :

```
domainname MonDom        # Ou ypdomainname (confusion avec .univ-lyon1.fr)
cd /usr/etc/yp           # Ou équivalent
./ypinit -m              # Création des Maps
vi /etc/?rc?             # Lancement de ''domainname'' ''ypserv'' ''ypbind''
```

Sur chacun des serveurs secondaire :

```
domainname MonDom        # Ou ypdomainname (confusion avec .univ-lyon1.fr)
cd /usr/etc/yp           # Ou équivalent
./ypinit -s maitre       # Création des Maps en les prenant du maître
vi /etc/?rc?             # Lancement de ''domainname'' ''ypserv'' ''ypbind''
```

Pour chacun des clients :

```
vi /etc/?rc?             # Lancement de ''domainname'' ''ypbind''
```

Les fichiers propagés sont les suivants (entre autre) :

- `/etc/passwd`
- `/etc/group`
- `/etc/netgroup`
- `/etc/hosts`
- `/etc/rpc`
- `/etc/services`

### 12.1.2 Comment utiliser NIS

Certain des fichiers remplacent les fichiers existants en pouvant créer des catastrophess: `services`, `hosts`, `rpc` Il est important d'avoir fusionné ces fichiers auparavant

Pour les password et les groupes, on peut directement inclure complètement le fichier du serveur en terminant le fichier `passwd` par `+::0:0:::` et le fichier `group` par `+:`

Evidemment ceci n'est possible que s'il n'y a pas de conflits entre les utilisateurs locaux et ceux du serveur. Il est difficile d'installer NIS après coups s'il existe déjà de nombreux utilisateurs.

Il est aussi possible de faire des groupes d'utilisateurs ou de machines ou de couples et donner des droits d'accès à l'un de ces groupes. Voici une partie de `/etc/netgroup`

```
m_bat m_ens m_rec m_ci
m_ci (ci6000.univ-lyon1.fr,-,) (ciprime.univ-lyon1.fr,-,)
ligia (-,pagnotte,) (-,manager,) (-,jmm,) permligia thesligia projligia
dealigia (-,foufou,) (-,olivier,) (-,blinski,) (-,sfoucher,) (-,tchich,)
```

On peut alors être sélectif dans le `/etc/passwd`

```
+@outils::0:0:::
+@rchrch::0:0:::
+@profs::0:0:::
```

Ou dans le `/etc/exports`

```
/disc1 m_bat
```

# 13 Exemples, réseau de serveurs, asynchrone

## 13.1 Mach OS

Machine virtuelle sur laquelle on construit des systèmes d'exploitation. Les entités gérées par `Mach` sont

- `Port`: Objet pouvant servir à envoyer et recevoir des messages mais aussi de capacité.
- `Task`: Un processus
- `Thread`: Une ligne d'exécution dans une `Task`
- `Message`: Un message. Il peut contenir un port d'authentification.
- `Memory`: Une zone mémoire

Les interruptions hard sont transformées en message vers le bon drivers

- De swaping si défaut de page.
- D'IO ou de cache si périphérique

Quelques `Thread` fonctionnent dans le noyau afin de faire quelques taches utiles : le ménage de ce qui n'est pas utilisé, le calcul de priorité, ...

Avantages de cette architecture :

- Le noyau est minimum : Micro kernel. De plus dans le cas de `Mach` il est multiprocesseur.
- Toutes les autres taches sont réalisées par d'autres serveurs.
- Plusieurs systèmes d'exploitation peuvent fonctionner en même temps sans que l'un émule l'autre.
- Le système d'exploitation n'est plus monolithique et donc plus facilement débuggable.
- Le système est plus lent car il est dispersé entre plusieurs processus, par contre il permet des communications directes en sautant des couches si c'est possible et donc finalement l'un compense l'autre.
- On sépare encore plus parties dépendantes du hardware des indépendantes.
- Des facilités pour créer son propre système d'exploitation.

Un exemple : `HURD`

- `Password`: Serveur permettant de valider des mots de passe.
- `Authentification`: Il associe un port par utilisateur connecté. Il ne donne le port qu'aux processus de l'utilisateur. Ceux-ci s'en servent pour prouver leur identité aux autres. Il associe une liste d'UID et PID à chaque utilisateur.
- `Exec`: Il remplace un exécutable par un autre.
- `Process`: Il associe des informations (que l'on veut lui donner) à chaque tache du système.
- `Terminal`: Il regroupe un certain nombre de tache et gère l'édition au clavier de lignes de commandes.
- `FileSystem`: Décode le support physique en fonction du système de fichier. Il peut être lancé pour tout utilisateur qui possède le fichier à décoder. Un `Port` par fichier ouvert.

Dans cette architecture, énormément de choses peuvent être faite sans faire appel au noyau mais à d'autres serveur. Baucoup d'informations peuvent passer directement d'un endroit à l'autre. D'autre part, la plupart des serveurs peuvent être remplacé par des versions des utilisateurs.

D'autre part, en ajoutant un serveur de `Port` sur chaque machine ont peut les faire communiquer de manière transparente sur le réseau.

## 13.2 IRC: Internet Relay Chat

RFC1459, commande “ircII”

IRC permet à des personnes de communiquer interactivement à travers le monde entier. Les communications peuvent avoir lieu entre utilisateurs ou bien d'un utilisateur vers tous les membres d'un canal de communication. Les canaux de communications sont créés par le seul fait de vouloir regarder dedans.

### 13.2.1 Organisation

Les *clients* sont les programmes intéractif qui permettent de dialoguer, il en existe sur tous les types de machines. Chaque *clients* se connecte sur un *serveur*.

Les *serveurs* gèrent les *clients* qui sont connecté sur eux ainsi que les messages venant des autres *serveurs* auquel il est connecté. Les messages s'échangeant entre serveurs presque identiques aux messages circulant entre client et serveur. On considère en fait que les autres serveurs sont des clients qui ont plus de droit que les autres.

Les messages et la syntaxe sont en ASCII lisible, cette méthode évite tout problème de codage machine.

L'organisation actuelle des serveurs est un *spanning tree*. Cette organisation simplifie la gestion du routage qui est alors trivial mais par contre crée des goulets d'étranglement.

### 13.2.2 Fonctionnement

Chaque serveur mémorise l'ensemble des serveurs auxquel il est connecté. Pour tous, il indique la branche sur lequel il est (une seule car c'est un arbre).

L'ensemble des serveurs mémorise un état courant. Tout serveur recevant une opération modifiant son état courant fait passer cette information aux serveurs auquel il est directement connecté (nouveau user, serveur, canal; destruction, ...)

En cas de rupture d'une liaison, on se retrouve avec 2 arbres séparés. Les 2 serveurs déconnecté envoient alors chacun de leur coté l'ensemble des commandes indiquant que plus rien n'existe de l'autre coté de la liaison. Il envoit donc des demandes de deconnexion de clients et serveurs. On se retrouve alors avec 2 arbres IRC différents fonctionnant chacun de leur coté.

Lors de la recréation d'une liaison, chacun réexplique à l'autre ce qu'il y a de son coté. On espère bien sur qu'il n'y aura aucun conflit. (C'est pour cela qu'il y a un réémanence de identificateurs) On évite aussi la création de cycle, en effet les connexions possibles qui créent des cycles sont invalidées jusqu'a ce qu'il soit peut-être interessant de la remttre en route.

### 13.2.3 Alors?

Ce réseaux de serveur contient une base de donnée globale mais qui est dupliquée sur chaque serveur...

L'arbre est un chemin de connexion trop restrictif.

C'est un magnifique exemple de clients serveur car un grand nombre de clients complètement différent peuvent se connecter au serveurs, pour avoir une communication vocal, pour jouer à des jeux (échecs, go, bridge, ...) il n'y a aucune limite.

# 14 Généralités

## 14.1 Agents

Un agent est un client à qui l'on a donné des instructions et qui interroge des serveurs afin de réaliser son objectif.

- Les `robots` d'indexation du WWW.

  Ce sont des programmes parcourant récursivement tous les liens qu'ils peuvent atteindre afin d'indexer leur contenu. Les robots qui ont le plus récupéré de document sont (début 1996) ceux de `Lycos`.

  Des fichiers d'instruction peuvent être laissé sur le serveur afin d'aider le robot dans sa recherche. Notamment éviter de lancer la récursion dans des documents infinis.
- Agents généraux du WWW

  Ce sont des programmes qui recherchent des documents dans le WWW en fonction de critères de recherche non pris en compte par les robot ds'indexation.

  Ces agents suivent les liens de manière intelligente afin de ne pas parcourir tous le WWW. Ils ont donc besoin d'une bonne base de mot-clef et de synonymes.
- Agents particuliers du WWW

  Un agent existe permettant d'accéder de manière transparente à l'ensemble des compagnies aériennes. Il présente donc à l'utilisateur des choix complet créés à partir de dizaines d'interrogations de serveurs différents

  Un autre vous permet de poster des nouvelles URL à tous les serveurs que cela intéresse. Celui-ci fonctionne en distant, votre agent ne tourne pas dans votre machine mais sur la machines qui héberge l'agent.