ANNEXE A4

Programmation Réseau (Syntaxe de quelques appels systèmes)

IP(4) **Manuel du programmeur Linux** **IP(4)**

NAME

ip - Implémentation Linux du protocole IPv4.

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

tcp_socket = socket(PF_INET, SOCK_STREAM, 0);
raw_socket = socket(PF_INET, SOCK_RAW, protocol);
udp_socket = socket(PF_INET, SOCK_DGRAM, protocol);
```

DESCRIPTION

Linux implémente le Protocole Internet (IP) version 4, décrit dans les RFC 791 et RFC 1122. ip contient une implémentation du multicasting niveau 2 conforme à la RFC 1112. Elle contient aussi un routeur IP comprenant un filtre de paquets.

L'interface de programmation est compatible avec les sockets BSD. Pour plus de renseignements sur les sockets, voir socket(7).

Une socket IP est créée par la fonction socket(2) invoquée sous la forme socket(PF_INET, type_socket, protocole). Les types valides des sockets sont SOCK_STREAM pour ouvrir une socket tcp(7), SOCK_DGRAM pour ouvrir une socket udp(7), ou SOCK_RAW pour ouvrir une socket raw(7) permettant d'accéder directement au protocole IP. Le protocole indiqué est celui inscrit dans les entêtes IP émis ou reçus. Les seules valeurs valides pour le protocole sont 0 et IPPROTO_TCP pour les sockets TCP, et 0 et IPPROTO_UDP pour les sockets UDP. Pour les sockets SOCK_RAW on peut indiquer un protocole IP IANA valide dont la RFC 1700 précise les numéros assignés.

Lorsqu'un processus veut recevoir de nouveaux paquets entrants ou connexions, il doit attacher une socket à une adresse d'interface locale en utilisant bind(2). Une seule socket IP peut être attachée à une paire (adresse, port) locale donnée. Lorsqu'on indique INADDR_ANY lors de l'attachement, la socket sera affectée à toutes les interfaces locales. Si on invoque listen(2) ou connect(2) sur une socket non affectée, elle est automatiquement attachée à un port libre aléatoire, avec l'adresse locale fixée sur INADDR_ANY.

L'adresse locale d'une socket TCP qui a été attachée est indisponible pendant quelques instants après sa fermeture, à moins que l'attribut SO_REUSEADDR ait été activé. Il faut être prudent en utilisant ce drapeau, car il rend le protocole TCP moins fiable.

FORMAT D'ADRESSE

Une adresse de socket IP est définie comme la combinaison d'une adresse IP d'interface et d'un numéro de port. Le protocole IP de base ne fournit pas de numéro de port, ils sont implémentés par les protocoles de plus haut-niveau comme udp(7) et tcp(7). Sur les sockets raw, le champ sin_port contient le protocole IP.

```
struct sockaddr_in {
  sa_family_t    sin_family;      /* famille d'adresses : AF_INET   */
  u_int16_t      sin_port;        /* port dans l'ordre d'octets réseau */
  struct in_addr sin_addr;        /* adresse Internet    */
};

struct in_addr {
  u_int32_t      s_addr;          /* Adresse dans l'ordre d'octets réseau */
};
```

sin_family est toujours rempli avec AF_INET. C'est indispensable. Sous Linux 2.2, la plupart des fonctions réseau renvoient EINVAL lorsque cette configuration manque. sin_port contient le numéro de port, dans l'ordre des octets du réseau. Les numéros de ports inférieures à 1024 sont dits réservés. Seuls les processus avec un UID effectif nul ou la capacité CAP_NET_BIND_SERVICE peuvent appeler bind(2) pour ces ports.
Notez que le protocole IPv4 en tant que tel n'a pas de concept de ports, ils sont seulement implémentés par des protocoles de plus haut-niveau comme tcp(7) et udp(7).

sin_addr est l'adresse IP de l'hôte. le membre addr de la structure struct in_addr contient l'adresse de l'interface de l'hôte, dans l'ordre des octets du réseau. in_addr ne doit être manipulé qu'au travers des fonctions de bibliothèque inet_aton(3), inet_addr(3), inet_makeaddr(3) ou directement par le système de résolution des noms (voir gethostbyname(3)). Les adresses IPv4 sont divisées en adresses unicast, broadcast et multicast. Les adresses unicast décrivent une interface unique d'un hôte, les adresses broadcast correspondent à tous les hôtes d'un réseau, et les adresses multicast représentent tous les hôtes d'un groupe multicast.

Les datagrammes vers des adresses broadcast ne peuvent être émis et reçus que si l'attribut de socket SO_BROADCAST est activé. Dans l'implémentation actuelle, les sockets orientées connexion ne sont autorisées que sur des adresses unicast.

Remarquez que l'adresse et le port sont toujours stockés dans l'ordre des octets du réseau. Cela signifie qu'il faut invoquer htons(3) sur le numéro attribué à un port. Toutes les fonctions de manipulation d'adresse et port de la bibliothèque standard fonctionne dans l'ordre du réseau.

Il existe plusieurs adresses particulières :
INADDR_LOOPBACK (127.0.0.1) correspond toujours à l'hôte local via le périphérique loopback ;
INADDR_ANY (0.0.0.0) signifie un attachement à n'importe quelle adresse ;
INADDR_BROADCAST (255.255.255.255) signifie n'importe quel hôte et à le même effet que INADDR_ANY pour des raisons historiques.

VOIR AUSSI

sendmsg(2), recvmsg(2), socket(4), netlink(4), tcp(4), udp(4), raw(4), ipfw(4)

SOCKET(2) Manuel du programmeur Linux SOCKET(2)

NOM

socket - Créer un point de communication.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

DESCRIPTION

Socket crée un point de communication, et renvoie un descripteur.

Le paramètre domain indique un domaine de communication, à l'intérieur duquel s'établira le dialogue. Ceci permet de sélectionner la famille de protocole à employer. Ces familles sont définies dans le fichier linux/socket.h.

Les formats actuellement proposés sont :

AF_UNSPEC	Famille non spécifiée, laisser le système la déterminer.
AF_UNIX	Protocoles locaux internes UNIX (pipe,...)
AF_INET	Protocoles Internet (UDP, TCP, etc...)
...	
AF_INET6	Réservé pour le projet IP version 6

Les sockets ont le type indiqué, ce qui fixe la sémantique des communications. Les types définis actuellement sont :

SOCK_STREAM	Support de dialogue garantissant l'intégrité, fournissant un flux de données binaires, et intégrant un mécanisme pour les transmissions de données hors-bande. Les sockets de ce type sont des flux full-duplex, similaires à des tubes.
SOCK_DGRAM	Transmissions sans connexion, non garantie, de datagrammes de longueur fixe, généralement courte.
SOCK_RAW	Transmissions internes au système, le type SOCK_RAW, ne peut être utilisé que par le Super-User.
SOCK_RDM	Transmission garantie de datagrammes
SOCK_SEQPACKET	Dialogue garantissant l'intégrité, pour le transport de datagrammes de longueur fixe. Le lecteur peut avoir à lire le paquet de données complet à chaque appel système read.

Le protocole à utiliser sur la socket est indiqué par l'argument protocol. Normalement il n'y a qu'un seul protocole par type de socket pour une famille donnée. Néanmoins rien ne s'oppose à ce que plusieurs protocoles existent, auquel cas il est nécessaire de le spécifier.

Le numéro de protocole dépend du domaine de communication de la socket. Voir protocols(5).

Une socket de type stream doit être connectée avant que des données puisse y être lues ou écrites. Une connexion sur une autre socket est établie par l'appel système connect(2). Une fois connectée les données y sont transmises par read(2) et write(2) ou par des variantes de send(2) et recv(2).

Quand une session se termine, on referme la socket avec close(2).

Les données hors-bande sont envoyées ou reçues en utilisant send(2) et recv(2).

Le protocole de communication utilisé pour implémenter les sockets stream garantit qu'aucune donnée n'est perdue ou dupliquée. Si un bloc de données, pour lequel le correspondant a suffisamment de place dans son buffer, n'est pas transmis correctement dans un délai raisonnable, la connexion est considérée comme inutilisable, et les appels systèmes renverront une valeur -1 en indiquant une erreur ETIMEDOUT dans la variable globale errno.

Eventuellement les protocoles peuvent maintenir les sockets en service en forçant des transmissions directes toutes les minutes en l'absence de toute autre activité. Une erreur est indiquée si aucune réponse n'est reçue sur une socket inactive pendant une période prolongée (par exemple 5 minutes).

Un signal SIGPIPE est envoyé au processus tentant d'écrire sur une socket inutilisable, forçant les programmes ne gérant pas ce signal à se terminer.

Les sockets de type SOCK_SEQPACKET emploient les mêmes appels systèmes que celles de types SOCK_STREAM, à la différence que la fonction read(2) ne renverra que le nombre d'octets requis, et toute autre donnée restante sera éliminée.

Les sockets de type SOCK_DGRAM ou SOCK_RAW permettent l'émission de datagrammes à des correspondants indiqués au moment de l'appel système send(2). Les datagrammes sont généralement lus par la fonction recvfrom(2), qui fournit également l'adresse du correspondant.

Un appel à fcntl(2) permet de préciser un groupe de processus qui recevront un signal SIGURG lors de l'arrivée de données hors-bande. Cette fonction permet également de valider des entrées/sorties non bloquantes, et une notification asynchrone des évènements par le signal SIGIO.

Les opérations sur les sockets sont représentées par des options du niveau socket. Ces options sont définies dans sys/socket.h. Les fonctions setsockopt(2) et getsockopt(2) sont utilisées respectivement pour fixer ou lire les options.

VALEUR RENVOYÉE

socket retourne un descripteur référençant la socket créée en cas de réussite. En cas d'échec -1 est renvoyé, et errno contient le code d'erreur.

ERREURS

EPROTONOSUPPORT	Le type de protocole, ou le protocole lui-même n'est pas disponible dans ce domaine de communication.
EMFILE	La table des descripteurs par processus est pleine.
ENFILE	La table des fichiers est pleine.
EACCES	La création d'une telle socket n'est pas autorisée.
ENOBUFS	Pas suffisamment d'espace pour allouer les buffers nécessaires.

BIND(2) Manuel du programmeur Linux BIND(2)

NOM

bind - Fournir un nom à une socket.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int  bind(int  sockfd, struct sockaddr *my_addr, int addrlen);
```

DESCRIPTION

bind fournit à la socket sockfd l'adresse locale my_addr. my_addr est longue de addrlen octets. Traditionnellement cette operation est appelée "assignation d'un nom à une socket" (Quand une socket est créée, par l'appel-système socket(2),elle existe dans l'espace des noms mais n'a pas de nom assigné).

NOTES

Assigner un nom dans le domaine UNIX crée une socket dans le système de fichiers, qui devra être détruite par le créateur une fois qu'il n'en a plus besoin, en utilisant unlink(2).

Les règles d'assignation de nom varient suivant le domaine de communication. Consultez le manuel du programmeur Linux section 4 pour de plus amples informations.

VALEUR RENVOYÉE

bind renvoie 0 s'il réussit, ou -1 s'il échoue, auquel cas errno contient le code d'erreur.

ERREURS

EBADFF	sockfd n'est pas un descripteur valide.
EACCESS	L'adresse est protégée et l'utilisateur n'est pas le Super-User.
ENOTSOCK	L'argument est un descripteur de fichier, pas une socket.
EADDRINUSE	La socket a déjà une adresse assignée.

Les erreurs suivantes sont spécifiques au domaine UNIX(AF_UNIX):

EINVAL	La longueur addr_len est fausse, ou la socket n'est pas de la famille AF_UNIX.
EROFS	L'i-noeud de la socket se trouverait dans un système de fichiers en lecture seule.
EFAULT	my_addr pointe en dehors de l'espace d'adresse accessible.
ENAMETOOLONG	my_addr est trop long
ENOENT	Le fichier n'existe pas.
ENOMEM	pas assez de mémoire pour le noyau.
ENOTDIR	Un composant du chemin d'accès n'est pas un répertoire.
EACCES	L'accès à un composant du chemin d'accès n'est pas autorisé.
ELOOP	my_addr contient des références circulaires (à travers un lien symbolique).

SEND(2) Manuel du programmeur Linux SEND(2)

NOM

send, sendto, sendmsg - Envoyer un message sur une socket.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int send(int s, const void *msg, int len , unsigned int flags);
int sendto(int s, const void *msg, int len , unsigned int flags,
            const struct sockaddr *to, int tolen);
int sendmsg(int s, const struct msghdr *msg , unsigned int flags);
```

DESCRIPTION

Send, sendto, et sendmsg permettent de transmettre un message à destination d'une autre socket. Send ne peut être utilisé qu'avec les sockets connectée alors que sendto et sendmsg peuvent être utilisés tout le temps.

L'adresse de la cible est donnée par to avec la longueur tolen. La longueur du message est indiquée dans len. Si le message est trop long pour être transmis intégralement au protocole sous-jacent, l'erreur EMSGSIZE sera déclenchée et rien ne sera émis.

Aucune indication d'échec de distribution n'est fournie par send. Seules les erreurs locales sont détectées, et indiquées par une valeur de retour -1.

Si la socket ne dispose pas de la place suffisante pour le message, alors send va bloquer, à moins que la socket ait été configurée en mode d'entrées/sorties non-bloquantes. On peut utiliser l'appel système select(2) pour vérifier s'il est possible d'émettre des données.

Le paramètre flags peut contenir une ou plusieurs des options suivantes

```
#define  MSG_OOB          0x1 /* Traiter les données hors-bande */
#define  MSG_DONTROUTE    0x4 /* Contourner le routage        */
```

L'option MSG_OOB est utilisée pour émettre des données hors-bande sur une socket qui l'autorise (par ex : SOCK_STREAM). Le protocole sous-jacent doit également autoriser l'émission de données hors-bande.

MSG_DONTROUTE est utilisée par les programmes de diagnostique ou de routage.
Voir recv(2) pour une description de la structure msghdr.

VALEUR RENVOYÉE

Ces appels systèmes renvoient le nombre de caractères émis, ou -1 s'ils échouent, auquel cas errno contient le code d'erreur.

ERREURS

EBADF	Descripteur de socket invalide.
ENOTSOCK	L'argument s n'est pas une socket.
EFAULT	Un paramètre pointe en dehors de l'espace d'adressage accessible.
EMSGSIZE	La socket nécessite une émission intégrale du message mais la taille de celui-ci ne le permet pas.
EWOULDBLOCK	La socket est non-bloquante et l'opération demandée bloquerait.
EPIPE	L'écriture est impossible (correspondant absent), et le signal SIGPIPE a été ignoré par le processus.
ENOMEM	Pas assez de mémoire pour le noyau.
ENOBUFS	La file d'émission de l'interface réseau est pleine. Ceci indique généralement une panne de l'interface réseau, mais peut également être dû à un engorgement passager.

RECV(2) Manuel du programmeur Linux RECV(2)

NOM

recv, recvfrom, recvmsg - Recevoir un message sur une socket.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int recv(int s, void *buf, int len, unsigned int flags);
int recvfrom(int s, void *buf, int len, unsigned int flags,
                struct sockaddr *from, int *fromlen);
int  recvmsg(int  s,  struct  msghdr  *msg,  unsigned  int flags);
```

DESCRIPTION

recvfrom et recvmsg sont utilisés pour recevoir des messages depuis une socket s, et peuvent servir à la lecture de données que la socket soit orientée connexion ou non.
Si from est non nul, et si la socket n'est pas orientée connexion, l'adresse de la source du message y est insérée. fromlen est un paramètre résultat, initialisé à la taille du buffer from, et modifié en retour pour indiquer la taille réelle de l'adresse enregistrée.

L'appel recv est normalement utilisé sur une socket connectée (voir connect(2)) et est équivalent à recvfrom avec un paramètre from nul. Comme ceci est redondant il est possible que recv ne soit plus maintenu dans le futur.

Ces trois routines renvoient la longueur du message si elles réussissent. Si un message est trop long pour tenir dans le buffer, les octets supplémentaires peuvent être abandonnés suivant le type de socket utilisé (voir socket(2)).

Si aucun message n'est disponible sur la socket, les fonctions de réception se mettent en attente, à moins que la socket soit non bloquante (voir fcntl(2)) auquel cas la valeur -1 est renvoyée, et errno est positionnée à EWOULDBLOCK.

Les fonctions de réception renvoient normalement les données disponibles dans la limite du paramètre len sans attendre d'avoir reçu le nombre exact réclamé. Ce comportement peut être modifié avec les options de socket SO_RCVLOWAT et SO_RCVTIMEO décrites dans getsockopt(2). L'appel select(2) peut permettre de déterminer si des données supplémentaires sont disponibles.

L'argument flags de l'appel recv est constitué par un OU binaire (|) entre les valeurs suivantes

MSG_OOB	traiter les données hors-bande
MSG_PEEK	lire sans enlever les données
MSG_WAITALL	attendre le nombre exact ou une erreur

L'option MSG_OOB permet la lecture des données hors-bande qui ne seraient autrement pas placées dans le flux de données normales. Certains protocoles placent ces données hors-bande en tête de la file normale, et cette option n'a pas lieu d'être dans ce cas.

L'option MSG_PEEK permet de lire les données en attente dans la file sans les enlever de cette file. Ainsi une lecture ultérieure renverra à nouveau les mêmes données.

L'option MSG_WAITALL demande que l'opération de lecture soit bloquée jusqu'à ce que la requête complète soit satisfaite. Toutefois la lecture peut renvoyer quand même moins de données que prévu si un signal est reçu, ou si une erreur ou une déconnexion se produisent.

VALEUR RENVOYÉE

Ces fonctions renvoient le nombre d'octets reçus si elles réussissent, ou -1 si elles échouent, auquel cas errno contient le code d'erreur.

ERREURS

EBADF	L'argument s n'est pas un descripteur valide.
ENOTCONN	La socket est associée à un protocole orienté connexion et n'a pas encore été connectée (voir connect(2) et accept(2)).
ENOTSOCK	L'argument s ne correspond pas à une socket.
EWOULDBLOCK	La socket est non-bloquante et aucune donnée n'est disponible, ou un délai de timeout a été indiqué, et il a expiré sans que l'on ait reçu quoi que ce soit.
EINTR	Un signal a interrompu la lecture avant que des données soient disponibles.
EFAULT	Un buffer pointe en dehors de l'espace d'adressage accessible.

Annexe 5

Les exceptions en C++

I – Le mécanisme try...catch...throw

La complexité de plus en plus grandes des applications informatiques, impliquant plusieurs développeurs, des bibliothèques de bas ou de haut niveau, des DLLs (ou équivalent) conduit parfois à des situations anormales ou exceptionnelles lors de l'utilisation (souvent par le client) de l'application. Les composants de bas niveau voient cette situation, mais ne savent pas à qui ni comment rendre compte, et ne peuvent ni ne doivent pas non plus décider de la conduite à tenir.

Il y a quelques années, ces situations se traduisaient par des plantages, des coups de téléphones et des dépannages d'urgence via des patches ou des versions « vite corrigées » des logiciels.

Pourtant, l'auteur des fonctions de haut niveau connaît à priori la manière de réagir aux situations anormales, mais ne peut pas toujours les anticiper ni les détecter. Le langage C++ (comme de nombreux autres langages modernes) met en place un « **mécanisme des exceptions** » permettant de manière générale à des fonctions appelées de notifier aux fonctions appelantes l'avènement d'une erreur interne.

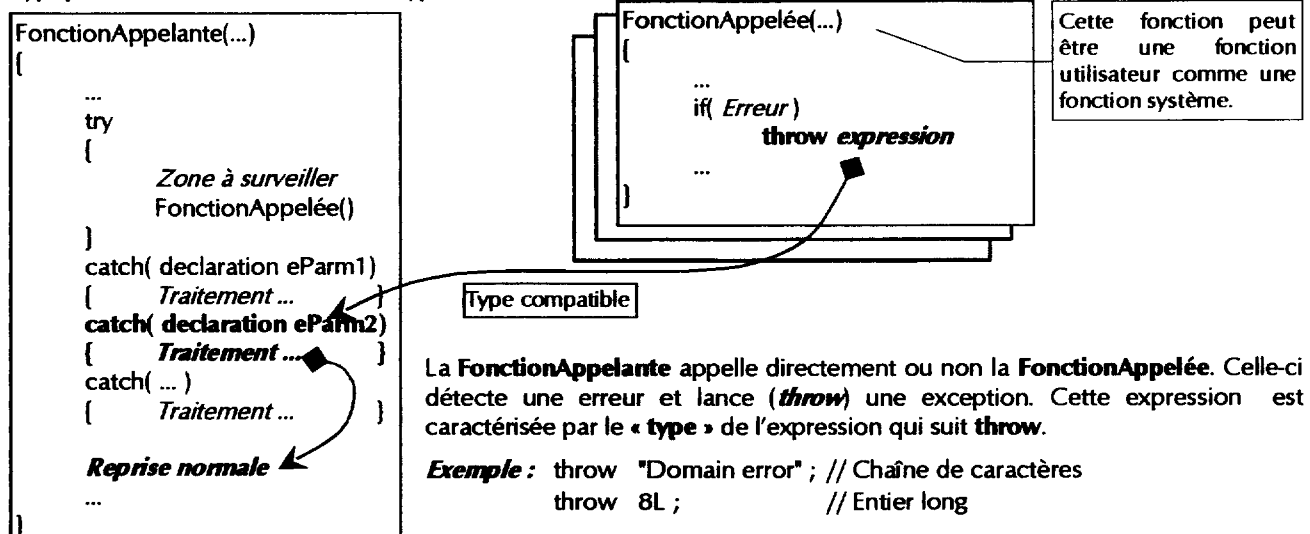
I-1 : Le fonctionnement du mécanisme

Les caractéristiques de ce mécanisme sont les suivantes :

- La fonction appelée qui détecte un événement exceptionnel construit une exception et la « lance » (*throw*) vers la fonction appelante ;
- Une exception est une instance (objet) d'une classe spécialement définie dans ce but (souvent une classe dérivée de la classe exception) comportant diverses informations utiles à la caractérisation de l'événement à signaler ; Ce peut être un nombre ou une chaîne.
- une fois lancée, l'exception traverse la fonction qui l'a lancée, sa fonction appelante, la fonction appelante de la fonction appelante, etc., jusqu'à atteindre une fonction active (c.-à-d. une fonction commencée et non encore terminée) qui a prévu d' « attraper » (*catch*) ce type d'exception ;
- lors du lancement d'une exception, la fonction qui l'a lancée et les fonctions que l'exception traverse sont immédiatement terminées : les instructions qui restaient à exécuter dans chacune de ces fonctions sont abandonnées ; malgré son caractère prématuré, cette terminaison prend le temps de détruire les objets locaux de chacune des fonctions ainsi avortées ;
- si une exception arrive à traverser toutes les fonctions actives, car aucune de ces fonctions n'a prévu de l'attraper, alors elle produit la terminaison du programme.

Pour intercepter des exceptions au sein d'un bloc d'instructions, une fonction doit délimiter cette séquence d'instructions au sein d'un bloc fonctionnel **try** d'une part, et d'autre part doit associer un traitement à chaque exception qu'elle souhaite intercepter à l'aide d'un gestionnaire **catch**.

Typiquement on a une structure du type :



L'exécution de la fonction appelée s'interrompt immédiatement après le **throw**, mais proprement : les destructeurs des objets locaux sont appelés avant de quitter la fonction.

On remonte désormais dans la pile des appels jusqu'à ce que les conditions suivantes soient vérifiées :

- L'appel de fonction ayant généré l'erreur a été fait depuis un bloc surveillé (bloc **try**).
- Un des blocs de traitement **catch** possède un paramètre compatible avec le type de l'expression associée au **throw**, c'est-à-dire :
    - De même type strictement, sans casting implicite.
    - La classe du **throw** est une classe dérivée de celle du **catch**.
- La fonction possède un bloc de type **catch(...)** qui intercepte toutes les exceptions.

Nota : Si aucune des fonctions - jusqu'au **main()** - n'intercepte l'exception, le programme s'arrête.

L'exécution reprend alors normalement à la suite de la dernière zone **catch**. Le code non exécuté de toutes les fonctions traversées est ignoré. L'exception n'est pas transmise, une fois traitée, aux fonctions appelantes. Il est cependant possible de relancer cette exception en réutilisant le mot-clé **throw** au sein des blocs **catch**, sans même renommer l'exception (on écrit simplement **throw ;**).

I-2 : Préciser le prototype des fonctions et méthodes

Le prototype **void Fonction( ) throw(int) ;** amène une précision quand au fonctionnement de la fonction. Celle-ci ne peut émettre (laisser échapper) que des exceptions de type **int**. D'autres exemples :

- **void Fonction( ) throw(...) ;** est susceptible de laisser échapper n'importe quelle exception.
- **void Fonction( ) throw( ) ;** ne laisse normalement échapper aucune exception.
- **void Fonction( ) throw(int , float) ;** peut laisser échapper des exceptions de type **int** ou **float**.

II – La classe « exception »

Le langage C++ définit l'existence d'une classe de base dont la déclaration est :

```
class exception
{
  public:
       exception() throw();
       exception(const exception &e) throw();
       exception &operator=(const exception &e) throw();
       virtual ~exception() throw();
       virtual const char *what() const throw();
};
```

Note : La réalité est malheureusement un tout petit peu plus compliquée...
Cette classe est bien une classe de base, et ne devrait jamais être utilisée hors du mécanisme de dérivation. En particulier, la méthode **what()** de la classe **exception** renvoie le message « **Unknown exception** ». La librairie standard dérive elle-même cette classe en de nombreuses versions.

III - Exemple

La classe Vector possède un constructeur qui prend en paramètre le nombre de composantes du vecteur. 0 est une valeur interdite qui lève une exception.

```
class Vector
{
  public :
      int *m_Datas ;
      ...
      Vector ( int Size) throw (exception)
      {
             if( Size ) m_Datas = new int [Size] ;
             else throw new exception ;
      }
      ...
};
```

Un programme appelant pourrait inclure les quelques lignes :

```
int main()
{
    try                    { Vector V( rand() % 100) ; }
    catch(exception e)     { cout << e.what() << " : pas de bol, hein?" ; }
}
```