Cours 4: Annexe

1

Annexe : Primitives

- ***Définition D'une Socket***

» création **s = socket (domaine, type, protocole)**

- domaine

» pf_unix : locale au système, nom de fichier dans l'arborescence
» pf_inet : accès au réseau en utilisant le protocole ip
» pf_route : passage de paramètres au noyau (tables de routages, table artp)

- type
  - » sock_stream : protocoles de type mode connecté (tcp)
  - » sock_dgram : protocoles de type mode datagramme (udp)
  - » sock_raw : utilisation directe des protocoles de bas niveau 3 (ip, icmp)
- protocole : identification du protocole utilisé. si 0, le système déduit ce champ des 2 paramètres précédents.

2

» adressage

```
struct sockaddr {
    u_char      sa_len ;        /*longueur totale*/
    u_char      sa_family ;     /*famille d_adresse*/
    char        sa_data [14];   /*valeurs */
} ;
```

- dans le fichier source <sys/socket.h>

```
struct sockaddr {
    u_int8_t        sin_len ;      /* longueur totale*/
    u_int8_t        sin_family ;  /*famille d_adresse*/
    u_int16_t       sin_port ;
    struct          in_addr   sin_addr ;
```

3

```
    int8_t          sin_zero[8];
    } ;
```

- dans le fichier source netinet/in.h

» lien entre la socket et le protocole

error = bind(s, adr, ,adrlong)

- error : entier qui contient le compte-rendu de l'instruction

» 0 : opération correctement déroulée

» -1 : une erreur est survenue

- s descripteur de la socket
- adr pointeur vers la zone contenant l'adresse de la station
- adrlong longueur de la zone adr

» ouverture d'une connexion

error = connect (s, destaddr, adrlong)

4

- mêmes paramètres que pour bind, la structure destadr contient l'adresse de la machine distante ainsi que le numéro de port à atteindre

» primitives du serveur

listen (s,backlog)

- permet de créer une file d'attente pour recevoir les demandes de connexion qui n'ont pas encore été prises en compte.

» s descripteur de la socket

» backlog nombre de requêtes maximum autorisées.

snew =accept (s, clientaddr, clientaddrlen)

- blocage dans l'attente d'une connexion si **accept** est ok
- les données peuvent être lues ou écrites à travers la socket snew

5

- *adresses locales et distantes d'une socket*

getsockname (s, localadr, addrlen)

- ermet de connaître l'adresse locale d'une socket (celle du bind).
- utile pour connaître le numéro de port attribué dynamiquement par le système.

getpeername (s, destadr, addrlen)

- permet de connaître l'adresse distante d'une socket (celle du connect) pour les sockets en mode connecté.

6

- *réception de données*

cc = read (s , buffer, taillemax)

- buffer est un pointeur vers la zone de réception.
- cc : nombre d'octets réellement reçus.

cc = recv (s , buffer, taillemax, drapeau)

- drapeau permet de configurer la connexion
- msg_oob : lecture "out of band" des messages urgents
- msg_peek : lecture des données sans les retirer du tampon

cc = recvfrom (s , buffer, taillemax, drapeau , émetteur, adrlg)

émetteur contient l'adresse de l'émetteur, utilisé en mode datagramme

7

- *émission de données*

write (s , buffer, longueur)

- utilisable uniquement en mode connecté (pas d'adresse de destinataire)

send (s , buffer, taillemax, drapeau)

- drapeau permet de configurer la connexion

» msg_oob : écriture "out of band" des messages urgents
» msg_dontroute : déboguage .

sendto (s , buffer, taillemax, drapeau , récepteur, adrlg)

- récepteur contient l'adresse du destinataire, utilisé en mode datagramme
- *fin d'utilisation d'une socket*

close (s)

- *accès aux bases de données relatives aux sites*

struct hostent * gethostbyname (name)

- interrogation sur /etc/hosts, nis, dns
- la structure est définie dans netdb.h

struct hostent * gethostbyaddr (addr, len, type)

8

gethostname (name , namelen)

- permet de connaître le nom de la machine locale sur laquelle s'éxécute le programme.

getnetbyname (name), getnetbyaddr (netaddr, addrtype)

- le réseau sur lequel on travaille

getprotobyname (name), getprotobynumber (number)

⑩

- le protocole utilisé

getservbyname (name , proto), getservbyport (port, proto)

- le service utilisé

9

Sockets : les options

level	optname	get	set	Description	flag	type de données
IPPROTO_IP	IP_OPTIONS	•	•	option de l'entête IP		
IPPROTO_TCP	TCP_MAXSEG	•		donne la taille max d'un segment	tcp	int
	TCP_NODELAY	•	•	ne pas retarder l'envoi pour grouper des paquets		int
SOL_SOCKET	SO_DEBUG	•	•	permet des infos de debugging	•	int
	SO_DONTROUTE	•	•	utilise uniquement les adresses d'interface	•	int
	SO_ERROR	•		rend le status de l'erreur	•	int
	SO_LINGER	•	•	contrôle de l'envoi des données après close		struct linger
	SO_OOBINLINE	•	•	concerne la réception de données hors bande	•	int
	SO_RCVBUF	•	•	taille du buffer de réception		int
	SO_SNDBUF	•	•	taille du buffer d'envoi		int
	SO_RCVTIMEO	•	•	timeout de réception		int
	SO_SNDTIMEO	•	•	timeout d'emission		int
	SO_REUSEADDR	•	•	autorise la réutilisabilité de l'adresse locale		int
	SO_TYPE	•		fournit le type de socket		int

10

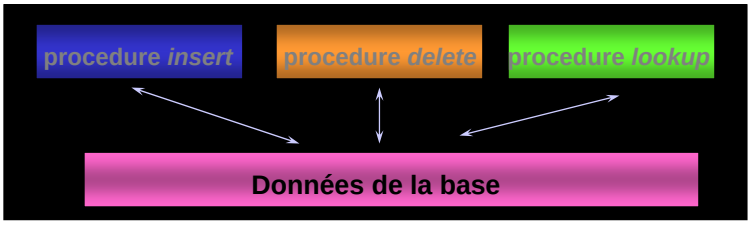
RPC : les apports de SUN

- Sun Microsystems a développé une technologie RCP dite « Sun RPC » devenue aujourd'hui un standard de fait; NFS (Network File Sytem) repose sur les RPC.
- Les Sun RPC définissent:
  - le format des messages que l'appelant (client) émet pour déclencher la procédure distante sur un serveur,
  - le format des arguments,
  - le format des résultats.
- Les protocoles UDP et TCP sont utilisés pour la communication et un protocole de présentation XDR assiste les RPC pour assurer le fonctionnement dans un environnement hétérogène.

11

RPC : le programme distant

- Programme distant : unité s'exécutant sur une machine distante.
- Un programme distant correspond à un serveur avec ses procédures et ses données propres.



Chaque programme distant est identifié par un entier unique codé sur 32 bits utilisé par l'appelant.

Les procédures d'un programme distant sont identifiées séquentiellement par les entiers 1, 2, ..., N.

12

- Une procédure distante est identifiée par le trio (prog, vers,proc)
  - *prog* identifie le programme distant,
  - *vers* la version du programme
  - *proc* la procédure.
- Groupe d'adresses pour programmes distants:

Nom	identifieur	description
portmap	100000	port mapper
rstat	100001	rstat, rup, perfmeter
ruserd	100002	remote users
nfs	100003	Network File System
ypserv	100004	Yellow pages (NIS)
mountd	100005	mount, showmount
dbxd	100006	debugger
ypbind	100007	NIS binder
etherstatd	100010	Ethernet sniffer
pcnfs	150001	NFS for PC

13

RPC : le programme distant

- Mode de communication RPC : «émission au moins une fois»:
  - si un appel de procédure distante s'exécutant sur UDP ne retourne pas, l'appelant ne peut pas savoir si la procédure a été exécutée ou si la réponse a été perdue.
  - De plus rien ne garantit que la procédure n'a été exécutée plusieurs fois suite à des duplications de la requête.
- La communication se fait en mode client/serveur: l'appelant spécifie l'adresse (IP, port) du serveur associé au programme distant.
- Problème sous-jacent: biunivocité adresse de port/identifieur de programme distant impossible (l'identifieur = 32 bits; port TCP ou UDP = 16 bits).

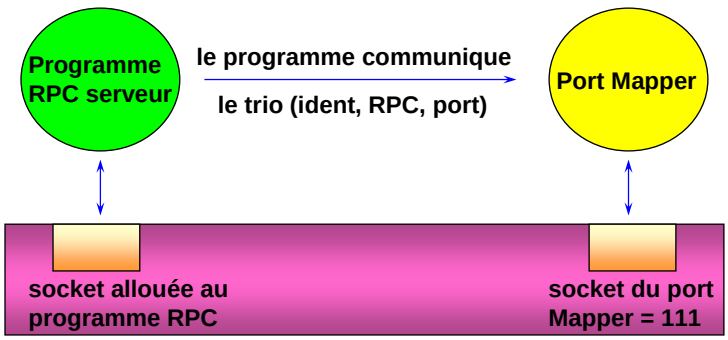
14

RPC : le programme distant

- Le problème de non biunivocité est résolu en allouant dynamiquement un numéro de port pour tout programme distant, et en le faisant connaître au client (appelant de RPC):
  - chaque machine offrant des programmes RPC dispose d'un service d'association de port dynamique: le *port mapper.*
  - Lorsqu'un programme RPC (serveur) démarre, il alloue dynamiquement un numéro de port local, puis contacte le *port mapper* de la machine sur laquelle il s'exécute, puis informe ce dernier de l'association (identifieur de programme RPC / numéro de port).
  - Le *port mapper* maintient une base de données renseignant les associations.
  - Lorsqu'un client désire contacter un programme RPC sur une machine M, il s'adresse au préalable au *port mapper* de M afin de connaître le port de communication associé.
  - Le *port mapper* s'exécute toujours sur le port de communication 111;

15

RPC : port mapper



16

RPC : eXternal Data Representation

- Le format général des messages RPC est de longueur variable, les champs des messages sont spécifiés dans le langage XDR (eXternal Data Representation).
- XDR : représentation des données définie par SUN Microsystems;
  - définit comment les données doivent être véhiculées sur un réseau.
  - permet d'échanger des données entre machines ayant des représentations internes différentes.
- Exemple : un entier de 32 bits ayant la valeur 260 sera représenté :
  - 0014 pour une machine de type « big endian » c'est à dire avec les MSB ayant les adresses basses et les LSB ayant les adresses hautes.
  - 4100 pour les machines « little endian ».

17

RPC : XDR

type	taille	description
int	32 bits	entier signé de 32 bits
unigned int	32 bits	entier non signé de 32 bits
bool	32 bits	valeur booléenne (0 ou 1)
enum	arb.	type énuméré
hyper	64 bits	entier signé de 64 bits
unsigned hyper	64 bits	entier non signé de 64 bits
float	32 bits	virgule flot. simple précision
double	64 bits	virgule flot. double précision
opaque	arb.	donnée non convertie
fixed array	arb.	tableau de longueur fixe de n'importe quel autre type
structure	arb.	agrégat de données
discriminated union	arb.	structure implémentant des formes alternatives
symbolic constant	arb.	constante symbolique
void	0	utilisé si pas de données
string	arb.	chaîne de car. ASCII

18

RPC : XDR

- L'encodage des données selon le format XDR contient uniquement les données représentées mais aucune information à propos du type de la donnée (contrairement au langage ASN1).
- Si une application utilise un entier de 32 bits, le résultat de l'encodage occupera exactement 32 bits et rien n'indiquera qu'il s'agit d'un type entier.
- Cette forme d'encodage implique que clients et serveurs doivent s'entendre sur le format exact des données qu'ils échangent.
- Une bibliothèque de fonction de conversion XDR permet aux concepteurs d'applications d'utiliser un logiciel standard sur tout type de machine.

19

RPC : XDR

Des fonctions de conversion XDR permettent aux concepteurs d'applications d'utiliser un logiciel standard sur tout type de machine.

```
XDR     *xdrs;      /* pointeur vers un buffer XDR*/
char    buf[BUFSIZE];   /* buffer pour recevoir les données encodées */
xdr_mem_create (xdr, buf, BUFSIZE, XDR_ENCODE);

/* maintenant un buffer stream est créé pour encoder les données
 * chaque appel à une fonction d'encodage va placer le résultat
 * à la fin du buffer stream; le pointeur sera mis à jour.
 */
int i;
```

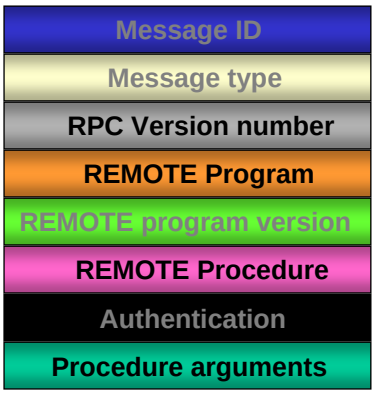
20

RPC : XDR

Fonction	arguments	type de donnée converti
xdr_bool	xdrs, ptrbool	booléen
xdr_bytes	xdrs, ptrstr, strsize, maxsize	chaîne de caractères
xdr_char	xdrs, ptrchar	caractère
xdr_double	xdrs, ptrdouble	virgule flot., double précision
xdr_enum	xdrs, ptrint	type énuméré
xdr_float	xdrs, ptrfloat	virgule flot. simple précision
xdr_int	xdrs, ip	entier 32 bits
xdr_long	xdrs, ptrlong	entier 64 bits
xdr_opaque	xdrs, ptrchar, count,	données non converties
xdr_bool	xdrs, ptrbool	booléen
xdr_bytes	xdrs, ptrstr, strsize, maxsize	chaîne de caractères
xdr_float	xdrs, ptrfloat	virgule flot. simple précision
xdr_int	xdrs, ip	entier 32 bits
xdr_long	xdrs, ptrlong	entier 64 bits
xdr_opaque	xdrs, ptrchar, count,	données non converties
xdr_pointer	xdrs, ptrobj	pointeur
xdr_short	xdrs, ptrshort	entier 16 bits
xdr_string	xdrs, ptrstr, maxsize	chaîne de caractères
xdr_u_char	xdrs, ptruchar	entier 8 bits non signé
xdr_u_int	xdrs, ptrint	entier 32 bits non signé

21

RPC : format des messages



Le format est de longueur variable car le nombre d'arguments de la procédure appelée ne peut être déterminé à l'avance

22

RPC : rpcgen

- Rpcgen est un outil de génération de logiciel produisant
  - le talon client,
  - un squelette de serveur,
  - les procédures XDR pour les paramètres et les résultats,
  - un fichier contenant les définitions communes.
- SUN fournit une méthodologie complète assistée par:
  - les routines de conversion XDR pour les types simples,
  - les routines XDR qui formatent les types complexes (tableaux et structures) utilisés dans la définition de messages RPC,
  - les fonctions run-time RPC qui permettent à un programme d'appeler une procédure distante, enregistrer un service auprès du *port mapper* , dispatcher une requête d'appel de procédure vers la procédure associée, à l'intérieur du programme distant; exemple de fonction run-time :

23

RPC : rpcgen

- La méthodologie consiste à développer l'application distribuée comme une application conventionnelle puis à définir les procédures qui seront exécutées à distance.
- Ce découpage implique l'adjonction de code entre l'appel de procédure et la procédure distante:

côté client : le nouveau code doit:

- **encoder les arguments,**
- **créer un message RPC CALL,**
- **émettre ce message vers le programme distant,**
- **attendre les résultats et décoder ces résultats selon la représentation interne de la machine locale.**

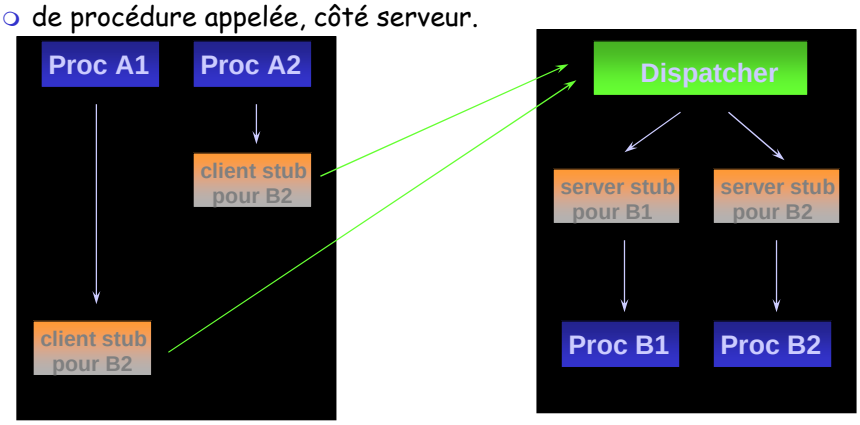
côté serveur : le nouveau code doit:

- **accepter une requête RPC,**
- **décoder les arguments selon la représentation de la machine locale,**
- **dispatcher le message vers la procédure adéquate,**
- **construire la réponse puis encoder celle-ci**
- **émettre le message correspondant vers le client.**

24

RPC : rpcgen

- Les procédures «stubs» remplacent les procédures conventionnelles:
  - d'appel de procédure, côté client,
  - de procédure appelée, côté serveur.



La procédure « stub » est dénommée exactement comme la procédure d'appel originale ce qui permet de conserver le même source dans la version distribuée; dans l'exemple ci-dessus la procédure « stub » pour B2 est appelée B2, la procédure « stub » pour B1 est appelée B1.

25

RPC : rpcgen

- Rpcgen génère automatiquement des portions de code nécessaires à l'implémentation d'une application distribuée.
- Cet outil utilise en entrée un fichier de spécification écrit par le concepteur d'application et génère en sortie les fichiers en langage C correspondants; il génère :
  - l'encodage des arguments,
  - l'envoi de message RPC,
  - le dispatching de procédure,
  - le décodage des données,
  - l'émission de réponse à un appel de procédure.
- Lorsqu'il est combiné avec les sources applicatives plus quelques fichiers écrits par le développeur, rpcgen génère entièrement les programmes client et serveur.

26

RPC : rpcgen

- Rpcgen sépare chaque procédure « stub » en deux parties :
  - une entité commune à toutes les applications qui consiste à la mise en oeuvre de la communication client/serveur,
  - une entité propre à chaque application, fournissant une interface avec celle-ci.
- Cette séparation est justifiée par le fait que rpcgen utilise les conventions d'appel de procédure du « package communication », tandis qu'il autorise l'utilisateur à utiliser les conventions d'appel des procédures distantes.

27