Bases de données
Forum

Sam Zoghaib
Tahina Ramananandro

5 juin 2005

1 Principe

1.1 Définition

Un *forum* est un système de communication informatique où des utilisateurs lisent et *postent* des *messages*, lisibles par tous. Les messages sont regroupés par thème, dans des *groupes de discussion*, ou en anglais *newsgroups*, ou, dans le jargon normalien, *contis*[1]. Un message peut appartenir à plusieurs contis, mais doit appartenir à au moins l'un d'eux. Mais les messages sont aussi organisés en arbres, grâce à une notion de *père-fils*, qui définit *un thread*, ou *fil de discussion*. Un thread peut s'étendre arbitrairement sur plusieurs contis. En plus de la notion de père-fils, il existe une notion plus générale de *référence* d'un message à un autre (comme une bibliographie), assez rarement utilisée dans un autre usage que la relation père-fils. (Le père d'un message est le dernier message référencé par celui-ci).

1.2 Motivation de notre système

Historiquement, les news ont commencé par le système de *mailing-list*. Dès 1970, les développeurs de divers composants du système *ITS*[1] (Incompatible Timesharing System) avaient mis en place de telles *mailing-lists*, pour recevoir les divers rapports de bugs.
Chaque message envoyé à une telle addresse était redirigé vers chaque personne inscrite.
De nombreux problèmes se posent. Notamment, chaque message étant stocké chez chaque utilisateur, une place importante est perdue ; tout le monde recevant les mêmes messages, il est plus judicieux de les stocker une seule fois et de donner aux utilisateurs les moyens de les lire depuis cet endroit.

En 1986 parait la RFC 977[2], qui décrit `NNTP`, protocole de news encore utilisé aujourd'hui, notamment sur U*SENET*. Ce protocole possède une grande flexibilité en centralisant les messages et permettant a divers serveurs de se « feeder » certains groupes (avoir des groupes en commun, chaque message posté dans un groupe sur un serveur étant copié sur l'autre). Toutefois, la RFC ne spécifie

[1] abrév. de *continuum*, c'est-à-dire ensemble de messages formant *continuité* de thème

pas de détails d'implémentation, par exemple la manière dont sont stockés les messages (base de données SQL, Berkeley DB, etc.). Il lui manque une fonction utile, celle de pouvoir aisément déplacer des messages d'un groupe a un autre. C'est pour pallier à ces défauts que nous avons choisi d'implémenter un serveur de news reposant sur une base de donnée SQL. Nous avons délibérement choisi de ne pas respecter la RFC 977, ce qui aurait demandé un travail beaucoup plus long et ne rentrant globalement pas dans le cadre de ce projet.

1.3 Structure d'un message

Un message est constitué d'*un en-tête*, suivi du *corps*. L'en-tête définit le message par des *attributs*, ou *champs*, de la forme `ATTRIBUT: valeur1, valeur2, ...`, tandis que le contenu du corps est libre. Il existe un certain nombre d'attributs obligatoires, sachant que tous ceux commençant par `X` sont toujours facultatifs. Les champs obligatoires dans notre système sont :

- `DATE` : la date du message
- `NEWSGROUPS` : l'ensemble des contis auxquels appartient le message.
- `FROM` : le pseudonyme du posteur
- `SENDER` : l'identité « physique » du posteur
- `SUBJECT` : le sujet du message
- `REFERENCES` : l'ensemble des messages auxquels ce message fait référence ; la dernière référence est le père du message.

Le message est toujours identifié par un identifiant unique, le *message-ID*.

1.4 Exemples

Exemples d'en-têtes de messages (l'attribut `DATE` est omis), ainsi que l'organisation correspondante en termes de contis et threads :

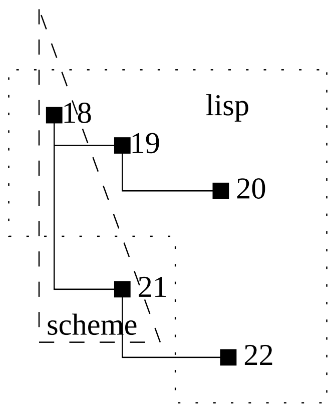
```
Message-Id: 18
NEWSGROUPS: lisp, scheme
FROM: zoghaib
SUBJECT: macros
REFERENCES:

Message-Id: 19
NEWSGROUPS: lisp
FROM: ramanana
SUBJECT: Re: macros
REFERENCES: 18

Message-Id: 20
NEWSGROUPS: lisp
FROM: zoghaib
SUBJECT: evalquote (Was: Re: macros)
REFERENCES: 18, 19

Message-ID: 21
NEWSGROUPS: scheme
FROM: ramanana
SUBJECT: Re: macros
REFERENCES: 18

Message-ID: 22
NEWSGROUPS: lisp
FROM: zoghaib
SUBJECT: Re: macros
REFERENCES: 19, 21
```



Le fait de poster un message contenant un nom de conti inconnu ne crée pas automatiquement ledit conti.

1.5 Modération : déplacement, ou blast

Entre le post, tâche commune, et la création de contis, tâche d'administration, il existe une tâche intermédiaire, dite de *modération*, visant à contrôler l'appartenance de messages à des contis en fonction de leur contenu. Cette tâche est accomplie par des *modérateurs*. Elle se manifeste par le *déplacement* de certains messages d'un conti à un autre, déplacement que l'on appelle aussi *blast* car les messages incriminés disparaissent de leur conti d'origine.

On peut blaster :

– un message isolé, depuis un conti vers un autre
– un fil entier, mais connexe dans un même conti (c'est-à-dire que le fil est « coupé » dès que l'on trouve un message fils hors du conti considéré), vers un autre conti

Mais le blast requiert des permissions spéciales (tout le monde ne peut pas blaster).

– Le modérateur qui blaste doit avoir la permission de déplacer des messages depuis le conti source considéré.
– Il existe des contis qui sont en *lecture seule*, c'est-à-dire que l'on ne peut pas

y poster. Un modérateur de ce conti peut toujours déplacer des messages depuis ce conti, mais n'a pas toujours le droit d'en faire un conti cible de blasts.

Prenons un exemple. Plaçons-nous dans la situation précédente, et supposons que l'on ait un troisième conti, `caml`. Supposons que `lisp` soit en lecture seule. Soient les deux modérateurs suivants :

Modérateur	`lisp`	`scheme`	`caml`
Alice	Oui, en lecture seule	Oui	Non
Bob	Oui, en lecture et écriture	Non	Non

Dans ces conditions, Alice peut déplacer le message 22 de `scheme` vers `caml`, mais pas vers `lisp` car il est en lecture seule. Ce que peut faire Bob, en revanche.

Toutefois, Alice, modératrice de `lisp`, peut, comme Bob, déplacer le message `18` de `lisp` vers `scheme`, même si Alice n'a pas le droit d'écriture sur `lisp` car ce droit ne s'applique que si le conti est la destination d'un blast.

Alice et Bob, modérateurs de `lisp`, peuvent choisir de déplacer tout le fil de `lisp` issu de `18`, qui comprendra donc les messages `18`, `19` et `20`, mais pas `22` car son père, `21`, n'est pas dans `lisp`.

Lorsqu'un message est déplacé d'un conti vers un autre, son Message-ID change.
C'est une convention utilisée dans la plupart des systèmes de news, dont NNTP, considérant qu'un message, déplacé, n'est plus le même message (en particulier si le Message-ID est un *hash* du contenu du message, en-tête et corps compris). Les clients news utilisant cette propriété pour savoir quels sont les messages lus ou non lus, il convient de ne pas la changer. En effet, ce serveur minimal étant fonctionnel, on pourrait vouloir l'étendre afin qu'il respecte la RFC 977. Ses fonctionnalités avancées nécessiteraient un client adapté pour être utilisé, mais les fonctionnalités de base (commune avec NNTP) doivent pouvoir être utilisées avec un client news usuel.

2 Modélisation en termes de bases de données

2.1 Schéma entité-association

La figure n'est pas prête.

2.2 Tables

Nous avons choisi de stocker le corps des messages dans des fichiers, et non dans la base de données. C'est ce que l'on appelle le *spool*.

Le code SQL de création des tables est donné ci-dessous :

```
create table contis (
        nom text primary key,
        description text,
        lecture_seule boolean default 'f' not null,
        prochain_numero integer default 0 not null
);
```

```
create table droits (
        moderateur text not null,
        conti text foreign key contis references nom,
        ecrire boolean,
        unique (moderateur,conti)
);

create table messages (
        id text primary key,
        pere text,
        fils text,
        subject text not null,
        sender text not null,
        "from" text not null,
        date text,
        numero_fichier integer default 0 not null
);

create table messages_contis (
        message text not null,
        conti text foreign key contis references nom,
        numero integer not null,
        unique (message, conti),
        primary key (conti, numero)
);

create table refs (
        de text not null,
        a text not null,
        unique (de, a)
);
```

La table `contis` définit les contis. L'attribut `prochain_numero` définit le numéro du prochain message qui appartiendra à ce conti (car outre par son Message-ID, on peut aussi accéder à un message par un conti auquel il appartient, et le numéro au sein de ce conti, défini dans l'attribut `numero` de la table `messages_contis`).
La table `droits` définit les modérateurs.
La table `messages` définit les messages. L'attribut `pere` contient le Message-ID du père, ou `NULL` si le message n'a pas de père.
L'attribut `fils` contient les Message-ID des fils du message, séparés par des virgules. L'attribut `numero_fichier` permet d'accéder au fichier spool correspondant au message (c'est le fichier qui contient le corps du message), dont le nom est `id.numero_fichier`.
La table `messages_contis` définit les appartenances de messages à des contis (on rappelle qu'un message peut appartenir à plusieurs contis). Comme un Message-ID peut changer, il n'est pas question d'intégrer ici pour l'attribut `message` une clé étrangère référençant `messages.id`, tout comme pour les attributs `de` et `a` de la table `refs`.
C'est dans cette table qu'est défini le numéro d'un message au sein d'un conti

(attribut `numero`). Cependant dans tous les contis auxquels il appartient, le message conserve son unique Message-ID.

Les tables sont utilisées :
- par le programme, pour le post et la modération
- directement, pour l'administration (création de contis et déclaration de modérateurs). Le rajout d'une interface pour la modération (démon ou interface web, par exemple) est triviale.

2.3 Instructions SQL pour l'administration

La création d'un conti se fait directement via l'instruction SQL suivante :
- si le conti est en lecture seule :

```
insert into contis(nom, lecture_seule)
   values ('nom_du_conti', 't');
```

- sinon, au choix :

```
insert into contis (nom)
   values ('nom_du_conti');
insert into contis (nom, lecture_seule)
   values ('nom_du_conti', 'f');
```

Un modérateur est ajouté également directement, par insertion de valeurs, sur le modèle de la création de conti.
- si le conti n'est pas en lecture seule

```
insert into droits (moderateur, conti)
   values ('nom_du_moderateur', 'nom_du_conti')
```

- si le conti est en lecture seule mais que l'on souhaite cependant que le modérateur ait le droit de déplacer des messages *depuis et vers* ce conti, on utilise au choix :

```
insert into droits (moderateur, conti)
  values ('nom_du_moderateur', 'nom_du_conti')
insert into droits (moderateur, conti, ecrire)
  values ('nom_du_moderateur', 'nom_du_conti', 't')
```

- si le conti est en lecture seule mais que l'on souhaite cependant que le modérateur ait le droit de déplacer des messages *uniquement depuis* ce conti :

```
insert into droits (moderateur, conti, ecrire)
  values ('nom_du_moderateur', 'nom_du_conti', 'f')
```

3 Le programme

Le programme a été écrit en C [3], avec la librairie `libpq` pour l'accès à la base de données PostgreSQL. Le module `base.c` concerne la base de données et le fonctionnement interne du forum, tandis que `server.c` est le code du démon qui permet d'utiliser le forum, notamment par le réseau. Notons toutefois que `base.c` utilise des extensions GNU de la Libc. Le code est raisonnablement portable, et devrait tourner, en ce qui concerne `server.c`, sur n'importe quel système POSIX, et en ce qui concerne `base.c`, sur n'importe quel système implémentant `asprintf()`.

3.1 Post

Le programme commence par générer des fragments de requêtes :

1. génère le Message-ID du message
2. vérifie, pour chaque message référencé, s'il existe, en exécutant[2] :

   ```
   select * from messages where id = '...'
   ```

   et génère la requête d'insertion correspondante dans la table des références :

   ```
   insert into refs(de,a)
     values ('nouveau_message','message_reference')
   ```

3. vérifie, pour chaque conti, s'il existe et s'il n'est pas en lecture seule, en exécutant :

   ```
   select * from contis where id = '...'
     and lecture_seule = 'f'
   ```

   et génère la requête d'insertion correspondante, avec l'incrément du numéro du prochain message :

   ```
   insert into messages_contis (message,conti,numero)
      values ('...', '...',
        (select prochain_numero from contis
           where nom = '...'));
   update contis
     set prochain_numero = 1 + prochain_numero
     where nom = '...';
   ```

   Si aucun conti ne convient, la création du message est annulée.
4. Sinon, le programme crée le fichier spool (contenant le corps du message), et génère la requête de création du message (`insert into messages...`).

Enfin, les différents fragments de requêtes générés sont remis dans l'ordre :

1. insertion du message
2. insertion des références
3. insertion dans les contis

puis exécutés au sein d'une même transaction, avec un verrou :

```
lock contis in row exclusive mode
```

au préalable pour éviter les conflits au niveau de l'incrémentation du compteur `prochain_numero`.

[2]La vérification consiste à compter le nombre de lignes renvoyées par l'exécution d'une telle requête. Si aucune ligne n'est renvoyée, la vérification échoue.

3.2 Déplacement d'un message

Lorsqu'un modérateur souhaite déplacer un message, le programme commence par vérifier si ce modérateur :

- est modérateur du conti source :

```
select * from droits where moderateur = '...' and conti = '...';
```

- a les droits d'écriture sur le conti cible s'il est en lecture seule :

```
select * from contis,droits
  where conti.nom = '...' and (
    conti.lecture_seule = 'f'
    or (
      droits.conti = conti.nom
      and droits.moderateur = '...'
      and droits.ecrire = 't'
    )
  )
```

(Si le conti n'est pas en lecture seule, cette requête, exécutée, renvoie toute la table `droits` jointe à la ligne de `contis` définissant le conti cible)

Si le modérateur passe ces vérifications avec succès, alors le message peut être déplacé. Le programme :

1. copie le fichier spool pour y mettre à jour l'en-tête `NEWSGROUPS`
2. supprime la ligne de `messages_contis` liant le message au conti source :

```
delete from messages_contis
  where message = 'id' and conti = 'conti_source'
```

3. vérifie si le message est déjà présent dans le conti cible :

```
select * from messages_contis
  where message = 'id' and conti = 'conti_cible'
```

   Si c'est le cas, le programme se contente d'incrémenter le compteur `numero_fichier` pour prendre en compte la copie du fichier spool, sans modifier le Message-ID, et la procédure est terminée :

```
update messages set numero_fichier = 1 + numero_fichier
  where message = 'id'
```

4. Si le message est absent du conti cible, alors il faut changer son Message-ID. Le programme génère alors un nouvel ID et renomme alors la copie du fichier spool dans ce sens,
5. sélectionne tous les messages faisant référence au message à déplacer :

```
select de from refs where a = 'ancien_id'
```

   puis pour chacun d'eux, crée une copie du fichier spool pour y mettre à jour l'en-tête `REFERENCES:`, et incrémente alors, comme ci-avant, le compteur `numero_fichier` (dans la table `messages`)
6. détecte l'éventuelle présence d'un père (en deux temps, vérifie d'abord si `pere` contient un Message-ID valide, puis teste l'existence du message),

pour mettre à jour l'attribut `fils` correspondant dans `messages` (attribut qui n'est pourtant pas utilisé outre mesure : cette étape pourrait donc très bien être ignorée et l'attribut `fils` supprimé dans les définitions des tables)

7. remplace alors par le nouvel ID toutes les occurrences de l'ancien parmi les fils (dans l'attribut `pere` de `messages`), ainsi que dans les attributs `de` et `a` de `refs` :

```
update messages set pere = 'nouvel_id'
  where pere = 'ancien_id';
update refs set de = 'nouvel_id'
  where de = 'ancien_id';
update refs set a = 'nouvel_id'
  where a = 'ancien_id';
```

8. modifie la ligne de définition du message dans `messages`, pour prendre en compte le nouvel ID :

```
update messages set id = 'nouvel_id',
  numero_fichier = 0
  where id = 'ancien_id'
```

9. et enfin, signale l'appartenance du message au conti cible :

```
insert into messages_contis (message, conti , numero)
    values ('nouvel_id', 'cible',
       (select prochain_numero from contis where
         nom = 'cible'));
update contis
 set prochain_numero = 1 + prochain_numero
  where nom = 'cible'
```

Notons que toutes ces requêtes sont exécutées dans une même transaction, avec au préalable des verrous amplement suffisants comme :

```
lock refs,messages,messages_contis in exclusive mode;
lock contis in row exclusive mode;
```

Ceci empêche le post de messages pendant l'opération de blast. En particulier, la réponse (par référence) à un message en train d'être déplacé (problème très désagréable en pratique : fils de discussion interrompus).

3.3 Déplacement d'un fil

On rappelle que, dans un fil, ne sont déplacés que :
- la racine, si elle appartient au conti source
- les messages appartenant au conti source et dont le père appartient au conti source

Avant toute chose, les vérifications des droits du modérateur sont effectuées une fois pour toutes.

Puis, une transaction unique est créée pour le déplacement de tout le fil. Le programme s'exécute alors récursivement, avec pour cas de base la racine du fil à déplacer :

1. Le programme déplace d'abord le message considéré du conti source vers le conti cible.
2. Puis, le programme cherche tous ses fils qui sont dans le conti source :

```
select messages.id from messages,messages_contis
  where messages.id = messages_contis.message
  and messages.pere = 'nouvel_id_du_pere'
  and messages_contis.contis = 'conti_source'
```

3. Puis, le programme déplace (récursivement, donc) le fil de discussion issu de chacun de ces messages, mais toujours depuis le même conti source, vers le même conti cible.

3.4 Utilisation du programme

Le programme est un serveur TCP qui écoute sur le port `SERV_TCP_PORT` défini dans serv.h et pouvant être spécifié au lancement du serveur, à l'aide de l'option `-p` ou `--port`. On interagit donc avec lui grâce à un programme de communication réseau, typiquement `telnet`.

Certains `#define` devraient être adaptés à la machine où l'on souhaite faire tourner le serveur avant la compilation. `serv.h` :

```
#define SPOOLPATH "chemin_de_spool"
#define PGCONN "host=hote_postgre␣dbname=base_de_donnees"
```

afin de spécifier le répertoire (à créer au préalable) où les fichiers spool seront créés, ainsi que les coordonnées de la base de données PostgreSQL contenant les tables (que l'on aura créées au préalable grâce aux instructions SQL du 2.2).

Les sources `base.c`, `server.c`, `serv.h` se compilent alors avec la librairie `libpq` : On conseille d'utiliser le `Makefile` fourni.

```
make
```

Une fois compilé, le programme `base` peut être lancé. Il se détache du terminal et redirige sa sortie standard et son erreur standard vers `/dev/null`, sauf si l'option `--debug` lui a été passée.

```
telnet host TCP_SERV_PORT
```

Le serveur répond alors :

```
Hey, how's it going ?
```

où l'on doit alors entrer `helo utilisateur`, sur quoi l'on reçoit le message :

```
Nice to meet you.
```

Là, on peut exécuter les commandes suivantes :

BLST messageid, source, cible	Déplace un message isolé. Attention à l'espace après chaque virgule.
BLTHR messageid, source, cible	Déplace le fil de discussion de source dont la racine est messageid. Attention à l'espace après chaque virgule.
GET message_id	Affiche le contenu du fichier spool du message dont l'ID est indiqué.
GETID conti, numero	Affiche le Message-ID correspondant au message indiqué par son conti et son numéro au sein du conti. Attention, bien respecter l'espace après la virgule.
LIST conti	Affiche la liste des messages (numéro et message-ID) dans le conti spécifié.
LSTNGS	Affiche la liste des contis.
POST	Poste un message. Sur les lignes suivantes, il faudra indiquer successivement et **dans l'ordre**, les champs NEWSGROUPS, FROM, SUBJECT et REFERENCES, sous la forme ATTRIBUT:<espace><contenu> (en plaçant aussi une espace après chaque virgule, si l'on indique plusieurs newsgroups ou plusieurs références). Ensuite, on pourra spécifier le corps du message, sur une ou plusieurs lignes, en terminant par une ligne blanche puis par une ligne ne contenant qu'un point.
QUIT	Coupe la connexion au serveur.

Notre choix d'avoir écrit un serveur permet en fait l'exécution de requêtes en parallèle depuis plusieurs machines, dans les conditions réelles d'utilisation d'un forum.
Le serveur peut être fermé en lui envoyant SIGTERM.

Références

[1] Donald Eastlake. ITS 1.5 Reference Manual. *MIT AI Lab Memo*, (161A), June 1968.

[2] Brian Kantor and Phil Lapsley. Network News Transfer Protocol. February 1986.

[3] Brian Kernighan and Dennis Ritchie. *The C Programming Language,* 2^{nd} *edition.* Prentice Hall, 1988.