Guide pratique du maintien de connexion TCP

Version française du TCP Keepalive HOWTO.

Fabio Busatto

`<fabio.busatto@sikurezza.org>`

Laurent Gauthier

Adaptation française `<laurent POINT mail CHEZ gmail POINT com>`

Eric Deschamps

Relecture de la version française `<erdesc CHEZ free POINT fr>`

Jean-Philippe Guérard

Préparation de la publication de la v.f. `<fevrier CHEZ tigreraye POINT org>`

Version : 1.0.fr.1.0

2007-05-04

Historique des versions		
Version 1.0.fr.1.0	2008-05-02	LG, ED, JPG
Première adaptation française.		
Version 1.0	2007-05-04	FB
Première édition, révisée par TM.		

Résumé

Ce document décrit l'implémentation du TCP keepalive dans le noyau linux, présente le concept global et détaille à la fois la configuration système et le développement d'application.

Table des matières

1. Introduction

Comprendre TCP keepalive n'est pas indispensable dans la plupart des cas, mais cela peut être très utile dans certaines circonstances. Il vous faudra posséder quelques notions de base des réseaux TCP/IP et de la programmation en langage C pour comprendre toutes les sections de ce document.

Le principal objectif de ce tutoriel (HOWTO) est de décrire en détail le TCP keepalive et de présenter différents cas d'application. Après avoir débuté avec un peu de théorie, le propos se concentre sur l'implémentation des routines TCP keepalive dans les noyaux Linux actuels (2.4.x, 2.6.x), et sur les moyens dont les administrateurs système peuvent tirer parti de ces routines, avec des exemples de configuration précis et des astuces.

La seconde partie de ce tutoriel met en jeu l'interface de programmation proposée par le noyau Linux, et le mode d'écriture des applications qui implémentent le TCP keepalive en langage C. Des exemples pratiques sont présentés, et une approche du projet `libkeepalive` est amorcée, permettant aux applications de bénéficier par héritage du keepalive sans modification de code.

1.1. Droits d'utilisation

1.2. Avertissement

Aucune responsabilité relative au contenu du présent document ne sera endossée. L'utilisation des concepts, exemples et informations est à vos propres risques. Des erreurs ou imprécisions peuvent endommager votre système. Agissez avec précaution, et même si cela est peu commun, l'auteur n’endosse aucune responsabilité (NdT : le traducteur non plus).

Tous les droits sont détenus par leurs propriétaires respectifs, sauf mention particulière. L’utilisation de termes de ce document ne doit pas être considérée comme une atteinte à la validité d’une marque déposée ou marque de service. Citer un produit ou une marque ne devrait pas être considéré comme répréhensible.

1.3. Remerciements et contributions

Ce travail ne doit à personne que je devrais remercier. Mais il doit à ma vie, et à mon savoir aussi: donc, merci à chacun m’ayant soutenu, avant ma naissance, actuellement, ou dans le futur. Sincèrement.

Un merci tout spécial à Tabatha, la femme patiente qui a lu mon travail et fait les corrections nécessaires.

1.4. Commentaires et corrections

Vos retours sur ce document seront les bienvenus. Adressez vos ajouts, commentaires et remarques à l’auteur à l’adresse mail suivante : `<fabio.busatto@sikurezza.org>`.

1.5. Traductions

Si vous êtes intéressé par la traduction de ce HOWTO en d’autres langues, n’hésitez pas à me contacter. Votre contribution sera la bienvenue.

Langues disponibles :

- anglais (document original)
- français

2. Aperçu de TCP keepalive

Afin de comprendre ce que fait TCP keepalive (que nous appellerons ’keepalive’), vous n’avez besoin que d’en lire le nom : keep TCP alive (maintenir TCP en vie), c’est à dire conserver la connexion TCP. Cela signifie que vous serez en mesure de vérifier l’état de votre socket de connexion (appelée aussi socket TCP), et de déterminer si la connexion est toujours établie ou si elle est rompue.

2.1. Qu’est-ce que TCP keepalive ?

Le concept du keepalive est très simple: lorsque vous initiez une connexion TCP, vous y associez un jeu de chronomètres. Certains de ces chronomètres ont trait à la procédure du keepalive. Quand la durée maximale du keepalive est atteinte, vous adressez à l’hôte distant un paquet sonde de keepalive ne contenant aucune donnée, avec le bit ACK positionné. Cela est possible grâce aux particularités de TCP/IP, une sorte de ACK doublé, et l’hôte distant n’aura aucun argument, puisque TCP est un protocole orienté flux. En retour vous aurez une réponse de l’hôte distant (qui n’a nul besoin d’implémenter le keepalive, mais seulement TCP/IP), sans donnée, et le ACK positionné.

Si vous recevez une réponse à votre sonde keepalive, vous pouvez être certain que la connexion est toujours établie et active sans inquiétude pour le niveau applicatif. Concrètement, TCP permet de maintenir un flux, sans paquet, donc un paquet de longueur zéro n'est pas dangereux pour le programme utilisateur.

Cette méthode est utile car si les autres points distants perdent leurs connexions (en raison d'un redémarrage par exemple) vous détecterez que la connexion est rompue, même sans avoir de flux de donnée. Si les sondes keepalive n'obtiennent pas de réponse, vous pouvez certifier que la connexion ne peut plus être considérée comme valide et agir en conséquence.

2.2. Pourquoi utiliser TCP keepalive ?

Vous pouvez vivre plutôt heureux sans keepalive, donc si vous lisez ces lignes, soit vous essayez de comprendre si keepalive est une réponse possible à vos problèmes, soit vous n'avez rien de plus intéressant à faire et c'est bien aussi. :)

Keepalive est non invasif, et dans la plupart des cas, si vous avez un doute, vous pouvez l'activer sans risque d'erreur. mais souvenez vous que c'est générateur de flux supplémentaire, ce qui peut avoir un impact sur les routeurs et les pare-feu.

En résumé, utilisez vos méninges et soyez prudent.

Dans la section suivante nous distinguerons les deux objectifs de keepalive:

- S'assurer qu'un hôte distant n'est pas injoignable
- Éviter une déconnexion due à une inactivité réseau.

2.3. Vérifier les hôtes injoignables

Keepalive peut être utilisé pour être averti que l'hôte distant est mort avant qu'il soit capable de vous le notifier. Cela pourrait se produire en différentes circonstances, une panique noyau ou une interruption soudaine du processus maintenant la connexion par exemple. Un autre cas justifiant keepalive pour détecter que l'hôte distant n'est pas joignable est la défaillance du réseau. Dans ce cas, si le réseau n'est pas à nouveau opérationnel, vous êtes dans la même situation que pour la mort de l'hôte distant. C'est dans ces cas de figure que les mécanismes TCP classiques ne permettent pas de s'assurer de l'état d'une connexion.

Songez à une simple connexion TCP entre l'hôte A et l'hôte B: il y a la poignée de main initiale en trois phases, le paquet SYN de A vers B, le SYN/ACK en retour de B vers A, et le ACK final de A vers B. A ce stade, nous sommes dans une situation stable : la connexion est établie, et les données peuvent donc être envoyées sur ce lien. Mais le problème survient : débranchez l'alimentation de B et instantanément il s'éteint, sans rien envoyer sur le réseau pour notifier A que la connexion va être interrompue. A, de son côté, est prêt à envoyer des données, et n'imagine pas que B est muet. Maintenant rebranchez l'alimentation de B et attendez que le système redémarre. A et B sont de retour, mais A présente une connexion toujours active vers B, alors que B l'ignore. La situation se résout d'elle-même lorsque A tente d'envoyer des données à B sur une connexion morte, et que B répond par un paquet RST, forçant A à finalement mettre fin à la connexion.

Keepalive peut vous notifier quand un destinataire devient injoignable sans risque de faux positif. En fait, si le problème tient au réseau entre les deux hôtes, le rôle du keepalive est d’attendre un temps pour tenter à nouveau, adressant le paquet keepalive avant de notifier de la rupture du lien.

```
 _____                                                        _____
|     |                                                      |     |
|  A  |                                                      |  B  |
|_____|                                                      |_____|
   ^                                                            ^
   |--->--->--->-------------- SYN -------------->--->--->---|
   |---<---<---<------------ SYN/ACK ------------<---<---<---|
   |--->--->--->-------------- ACK -------------->--->--->---|
   |                                                           |
   |                                   le système meurt ---> X
   |
   |                               le système redémarre ---> ^
   |                                                           |
   |--->--->--->-------------- PSH -------------->--->--->---|
   |---<---<---<-------------- RST --------------<---<---<---|
   |                                                           |
```

2.4. Éviter une déconnexion due à une inactivité réseau.

L’autre objectif utile de keepalive est d’éviter que l’inactivité ne provoque une déconnexion. C’est un cas fréquent d’être déconnecté sans raison lorsque vous vous trouvez derrière un proxy NAT ou un pare-feu. Ce comportement est dû aux procédures de surveillance des connexions des proxies et pare-feu, qui tiennent un inventaire des connexions qui les traverse. En raison des limites physiques de leurs ressources, ces machines ne peuvent conserver en mémoire qu’un nombre déterminé de connexions. La règle la plus courante et la plus logique est de maintenir les connexions les plus récentes et de mettre d’abord fin aux connexions les plus anciennes ou inactives.

Pour revenir à nos hôtes A et B, reconnectons les. Une fois le lien établi, attendons qu’un évènement se produise pour le transmettre à l’hôte distant. Qu’en est-il si cet évènement se produit après un long moment ? Notre connexion a sa propre durée, qui est inconnue du proxy. Lorsque nous finissons par transmettre des données, le proxy n’est plus capable de les traiter correctement, et la connexion est rompue.

Puisque le fonctionnement normal est de mettre en tête de liste la connexion par laquelle transitent des paquets, et de choisir la dernière connexion de la file quand il faut en supprimer une, le fait d’envoyer périodiquement des paquets sur le réseau est un bon moyen pour toujours rester en phase avec un risque minime de suppression.

```
 _____           _____                                        _____
|     |         |     |                                      |     |
|  A  |         | NAT |                                      |  B  |
|_____|         |_____|                                      |_____|
   ^               ^                                            ^
   |--->--->--->---|----------- SYN ------------->--->--->---|
   |---<---<---<---|--------- SYN/ACK -----------<---<---<---|
   |--->--->--->---|----------- ACK ------------->--->--->---|
   |               |                                           |
   |               | <--- connexion supprimée de la table      |
   |               |                                           |
   |--->- PSH ->---| <--- connexion invalide                   |
   |               |                                           |
```

3. Utiliser TCP keepalive sous Linux

Linux intègre nativement le keepalive. Vous devez activer le réseau TCP/IP pour pouvoir l'utiliser. Vous avez aussi besoin du support de `procfs` et de `sysctl` pour pouvoir configurer les paramètre noyau au lancement.

Les fonctions impliquant keepalive utilisent trois variables manipulées par l'utilisateur :

`tcp_keepalive_time`

intervalle entre le dernier envoi de paquet (le simple ACK n'étant pas considéré comme de la donnée) et la première sonde keepalive; après que la connexion ait été marquée comme requérant un keepalive, ce compteur n'est plus utilisé.

`tcp_keepalive_intvl`

intervalle entre deux sondes keepalive, indépendamment de ce qui est échangé sur la connexion dans l'intervalle

`tcp_keepalive_probes`

nombre de sondes non acquittées à envoyer avant de considérer la connexion perdue et de notifier la couche applicative.

Rappelez-vous que le support du keepalive, même s'il est configuré dans le noyau, n'est pas le comportement par défaut de Linux. Les programmes doivent requérir le contrôle du keepalive pour que ses sockets puissent utiliser l'interface `setsockopt`. Relativement peu de programmes implémentent keepalive, mais vous pouvez facilement ajouter le support du keepalive pour la plupart d'entre eux en suivant les instructions détaillées plus avant dans ce document.

3.1. Configurer le noyau

Il existe deux moyens de configurer les paramètres keepalive du noyau au travers de commandes utilisateur:

- l'interface `procfs`
- l'interface `sysctl`

Nous aborderons essentiellement comment procéder au travers de l'interface procfs car elle est la plus utilisée, recommandée et la plus simple à appréhender. L'interface sysctl, particulièrement sous l'aspect de l'appel système (syscall) `sysctl`(2) et non de l'outil **sysctl**(8), n'est là qu'à titre informatif.

3.1.1. L'interface`procfs`

Cette interface nécessite que`sysctl` et `procfs` soient inclus au noayu, et que `procfs` soit monté quelque part dans le système de fichiers (habituellement `/proc`, comme dans l'exemple ci-dessous). Vous pouvez lire les valeurs des paramètres actuels en listant avec la commande « cat » les fichiers du répertoire `/proc/sys/net/ipv4/` :

```
# cat /proc/sys/net/ipv4/tcp_keepalive_time
7200

# cat /proc/sys/net/ipv4/tcp_keepalive_intvl
75

# cat /proc/sys/net/ipv4/tcp_keepalive_probes
9
```

Les deux premiers paramètres sont exprimés en secondes, et le dernier est un nombre simple. Cela signifie que les routines keepalive attendent deux heures (7200 secs) avant d'adresser la première sonde keepalive, et en adressent une nouvelle toutes les 75 secondes. Si aucune réponse ACK n'est reçue après neuf tentatives consécutives, la connexion est considérée comme rompue.

La modification de ces valeurs est directe : il faut écrire de nouvelles valeurs dans les fichiers. Supposons que souhaitiez configurer la machine pour que le keepalive débute après dix minutes d'inactivité sur le lien, et que des sondes soient envoyées chaque minute. En raison de l'instabilité de ce brin de votre réseau et de la faible valeur de l'intervalle, supposons que vous vouliez augmenter le nombre de tentatives à 20.

Voici comment paramétrer ces valeurs :

```
# echo 600 > /proc/sys/net/ipv4/tcp_keepalive_time

# echo 60 > /proc/sys/net/ipv4/tcp_keepalive_intvl

# echo 20 > /proc/sys/net/ipv4/tcp_keepalive_probes
```

Pour confirmer la prise en compte des nouvelles valeurs, affichez à nouveau le contenu des fichiers pour vérifier qu'ils présentent bien les valeurs souhaitées.

Il faut garder présent à l'esprit que `procfs` manipule des fichiers spéciaux, et vous ne pouvez pas tout faire sur ces fichiers qui ne sont qu'une interface vers l'environnement du noyau, non de véritables fichiers. Testez vos scripts avant de les utiliser et faites en sorte d'utiliser des modes d'accès simples comme dans les exemples ci-dessus.

Vous pouvez accéder à l'interface grâce à l'outil **sysctl**(8), en précisant ce que vos voulez lire ou écrire.

```
# sysctl \
> net.ipv4.tcp_keepalive_time \
> net.ipv4.tcp_keepalive_intvl \
> net.ipv4.tcp_keepalive_probes
net.ipv4.tcp_keepalive_time = 7200
net.ipv4.tcp_keepalive_intvl = 75
net.ipv4.tcp_keepalive_probes = 9
```

Remarquez que les noms `sysctl` sont proches des chemins `procfs`. L'écriture se fait grâce à l'option `-w` de **sysctl** (8):

```
# sysctl -w \
> net.ipv4.tcp_keepalive_time=600 \
> net.ipv4.tcp_keepalive_intvl=60 \
> net.ipv4.tcp_keepalive_probes=20
net.ipv4.tcp_keepalive_time = 600
net.ipv4.tcp_keepalive_intvl = 60
net.ipv4.tcp_keepalive_probes = 20
```

Remarquez que **sysctl** (8) n'utilise pas l'appel système (syscall)`sysctl`(2), mais lit et écrit directement dans l'arborescence `procfs`, donc `procfs` devra être activé dans le noyau et monté dans le système de fichiers, comme si vous accédiez directement aux fichiers via l'interface `procfs`. **Sysctl**(8) n'est qu'un moyen différent de faire la même chose.

3.1.2. L'interface`sysctl`

Il existe un autre moyen d'accéder aux variables du noyau : l'appel système `sysctl` (2). Cela peut être utile lorque `procfs` n'est pas disponible du fait que la communication avec le noyau est réalisée directement via syscall et pas au travers de l'arborescence `procfs` . Il n'existe actuellement aucun programme qui implémente l'appel syscall (souvenez-vous que **sysctl**(8) ne l'utilise pas).

Pour une utilisation détaillé de `sysctl`(2) reportez vous au manuel (man).

3.2. Rendre les modifications persistantes au redémarrage

Il existe différents moyens de paramétrer le système à chaque démarrage. Tout d'abord, souvenez vous que chaque distribution Linux possède son propre jeu de scripts d'initialisation appelé par **init** (8). Les configurations les plus courantes incluent soit le répertoire `/etc/rc.d/` , soit `/etc/init.d/` . Dans ce cas vous pouvez positionner les paramètres dans un script de démarrage quelconque, keepalive relisant les valeurs à chaque fois que ses routines en ont besoin. Donc si vous changez la valeur de `tcp_keepalive_intvl` alors que la connexion est encore active, le noyau utilisera la nouvelle valeur pour continuer.

Les commandes d'initialisation peuvent logiquement être placées en trois endroits différents : le premier est dans la configuration réseau, le second dans le script `rc.local` , habituellement inclus dans toutes les distributions, et connu comme étant le point de configuration utilisateur au démarrage. Le troisième point existe peut-être déjà sur votre système. En revenant à l'outil **sysctl** (8) , vous pouvez voir que l'option `-p` charge les paramètres du fichier de configuration `/etc/sysctl.conf` . Il est fréquent que votre script d'initialisation exécute déjà **sysctl** `-p` (un « grep » sur le répertoire de configuration le confirmera), et vous n'avez alors qu'à ajouter les lignes dans `/etc/sysctl.conf` pour qu'elles soient prises en compte à chaque démarrage. Pour davantage d'informations sur la syntaxe de `sysctl.conf` (5), reportez vous au manuel.

4. Programmer des applications

Cette section aborde le code nécessaire à l'écriture d'une application utilisant keepalive. Ce n'est pas un manuel de programmation, et il requiert d'avoir une connaissance du langage C et des concepts réseau. Je considère que la notion de socket vous est familière, de même que tous les aspects généraux de votre application.

4.1. Quand votre code requiert keepalive

Les applications réseau ne requièrent pas toutes l'aide du keepalive. Souvenez vous qu'il s'agit de TCP keepalive. Comme vous pouvez le deviner, seules les sockets TCP peuvent en tirer parti.

La plus belle chose que vous puissiez faire en écrivant une application est de la rendre aussi paramétrable que possible, et de ne pas en forcer les choix. Si vous voulez prendre en compte le bonheur de vos utilisateurs, vous devriez implémenter keepalive et laisser les utilisateurs décider s'ils veulent ou non l'utiliser en prévoyant un paramètre de configuration ou une option de ligne de commande.

4.2. L'appel de fonction `setsockopt`

Tout ce dont vous avez besoin pour que keepalive soit activé sur une socket particulière est de positionner l'option sur cette socket. Le prototype de fonction est le suivant:

```
int setsockopt(int s, int level, int optname,
               const void *optval, socklen_t optlen)
```

Le premier paramètre est la socket, préalablement créée avec `socket(2)`; le second doit être `SOL_SOCKET`, et le troisième `SO_KEEPALIVE` . Le quatrième doit être un entier booleéen, indiquant que l'option est active, alors que le dernier est la taille de la valeur passée précédemment.

Conformément au manuel, le code retour 0 indique le succès, -1 est la valeur d'erreur (et `errno` est correctement renseignée).

Il existe aussi trois autres options de socket qu'il est possible de renseigner en écrivant votre application. Toutes utilisent le niveau `SOL_TCP` au lieu de `SOL_SOCKET`, et elles prennent le pas sur les variables système pour la socket courante. Si vous lisez avant d'écrire, les paramètres système seront retournés.

- `TCP_KEEPCNT` : prend le pas sur `tcp_keepalive_probes`
- `TCP_KEEPIDLE` : prend le pas sur `tcp_keepalive_time`
- `TCP_KEEPINTVL` : prend le pas sur `tcp_keepalive_intvl`

4.3. Exemples de code

Voici un petit exemple qui crée une socket, montre que keepalive est désactivé, puis l'active et vérifie que l'option est réellement positionnée.

```
            /* --- début du programme de test  KEEPALIVE --- */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(void);

int main()
{
   int s;
   int optval;
   socklen_t optlen = sizeof(optval);

   /* Creation de la socket */
   if((s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
      perror("socket()");
      exit(EXIT_FAILURE);
   }

   /* Verifie l'etat de l'option keepalive */
   if(getsockopt(s, SOL_SOCKET, SO_KEEPALIVE, &optval, &optlen) < 0) {
      perror("getsockopt()");
```

```
      close(s);
      exit(EXIT_FAILURE);
   }
   printf("SO_KEEPALIVE is %s\n", (optval ? "ON" : "OFF"));

   /* Rend l'option active */
   optval = 1;
   optlen = sizeof(optval);
   if(setsockopt(s, SOL_SOCKET, SO_KEEPALIVE, &optval, optlen) < 0) {
      perror("setsockopt()");
      close(s);
      exit(EXIT_FAILURE);
   }
   printf("SO_KEEPALIVE set on socket\n");

   /* Verifie a nouveau son etat */
   if(getsockopt(s, SOL_SOCKET, SO_KEEPALIVE, &optval, &optlen) < 0) {
      perror("getsockopt()");
      close(s);
      exit(EXIT_FAILURE);
   }
   printf("SO_KEEPALIVE is %s\n", (optval ? "ON" : "OFF"));

   close(s);

   exit(EXIT_SUCCESS);
}

            /* ---  fin du programme de test  KEEPALIVE  --- */
```

5. Implémenter keepalive sur une application tierce

Tout le monde n'est pas développeur d'application, et tout le monde ne réécrira pas entièrement une application pour combler le manque d'une fonctionnalité. Peut-être souhaitez vous ajouter keepalive à une application existante, et même si son auteur n'a pas considéré cela important, vous pensez que ce sera utile.

Tout d'abord, souvenez vous de ce qui a été dit précédemment à propos des cas où keepalive est nécessaire. Ensuite vous devrez affecter les sockets TCP orientées connexion.

Comme Linux ne fournit pas la possibilité d'activer le support keepalive via le noyau (les OS de type BSD le permettent souvent), le seul moyen est d'appeler `setsockopt` (2) après la création de la socket. Il y a deux solutions:

- modification du code source du programme original
- injection `setsockopt` (2) en utilisant la technique de préchargement de bibliothèque

5.1. Modifier le code source

Souvenez-vous que keepalive n'est pas orienté programme, mais orienté socket, donc si vous avez de multiples sockets, vous pouvez gérer keepalive séparément pour chacune d'entre elles. La première étape consiste à comprendre ce que fait le programme, la seconde à rechercher le code pour chaque socket dans le programme. Cela peut être fait en utilisant **grep**(1), comme suit:

```
# grep 'socket *(' *.c
```

Cela vous montrera à peu près toutes les sockets du code. L'étape suivante consiste à choisir les bonnes : vous ciblez les sockets TCP, donc recherchez PF_INET (ou AF_INET), SOCK_STREAM et IPPROTO_TCP (ou plus communément, 0) dans les paramètres de votre liste de sockets, et enlevez celles qui ne correspondent pas.

Il existe un autre moyen de créer une socket au travers de accept(2). En ce ce cas, suivez les sockets TCP identifiées et vérifiez si certaines sont en écoute : si c'est le cas, gardez à l'esprit que accept(2) retourne un descripteur de socket, qui doit être ajouté à votre liste de sockets.

Une fois les sockets identifiées, vous pouvez procéder aux modifications. Le patch le plus 'fast & furious' peut consister à simplement ajouter la fonction setsockopt(2) juste après le bloc de création de la socket. Éventuellement, vous pouvez ajouter des appels supplémentaires pour modifier les paramètres systèmes par défaut de keepalive. Surtout soyez attentif au positionnement des vérifications d'erreurs et des handlers de la fonction, peut-être en recopiant le style du code alentour. Songez à affecter à optval une valeur non nulle et à initialiser optlen avant d'appeler la fonction.

Si vous en avez le temps ou pensez que ce serait plus élégant, essayez d'implémenter complètement le keepalive à votre programme, en incluant une option de ligne de commande ou un paramètre de configuration pour laisser à l'utilisateur la liberté d'utiliser ou non keepalive.

5.2. libkeepalive: préchargement de bibliothèque

Dans de nombreux cas vous n'avez pas la possibilité de modifier le code source d'une application, ou bien lorsque vous devez activer keepalive pour tous vos programmes, tout patcher et tout recompiler n'est pas recommandé.

Le projet libkeepalive a vu le jour pour faciliter l'implémentation du keepalive au sein des applications puisque le noyau Linux ne permet pas de le faire nativement (comme le fait BSD). La page d'accueil du projet libkeepalive est disponible à l'adresse http://libkeepalive.sourceforge.net/

Il consiste en une bibliothèque partagée qui outrepasse l'appel système socket de la plupart des exécutables, sans aucun besoin de les recompiler ni de les modifier. La technique repose sur la fonctionnalité de pré-chargement (*preloading*) de **ld.so**(8), chargeur inclus dans Linux, qui qui permet le chargement de bibliothèques partagées avec une priorité supérieure à la normale. Les programmes utilisent habituellement l'appel de fonction socket (2) situé dans la glibc, librairie partagée; avec libkeepalive il est possible d'encapsuler la fonction setsockopt(2) juste après la création, retournant au programme principal une socket avec keepalive déjà positionné. En raison des mécanismes utilisés pour réaliser l'appel système, ce procédé ne fonctionne pas lorsque la fonction socket est compilée statiquement dans le binaire, comme dans le cas d'un programme lié par l'option -static de **gcc**(1).

Après avoir téléchargé et installé libkeepalive, vous serez en mesure d'ajouter le support de keepalive à vos programmes sans être root au préalable, simplement en initialisant la variable d'environnement LD_PRELOAD avant d'exécuter le programme. Au fait, le super utilisateur peut aussi forcer la pré-chargement au travers d'une configuration globale, et les utilisateurs peuvent choisir de le désactiver en positionnant la variable d'environnement KEEPALIVE à off.

L’environnement est aussi utilisé pour positionner des valeurs spécifiques pour les paramètres de keepalive, vous avez donc la possibilité de gérer chaque programme de façon distincte, en initialisant `KEEPCNT`, `KEEPIDLE` et `KEEPINTVL` avant de lancer l’application.

Voici un exemple d’utilisation de libkeepalive :

```
$ test
SO_KEEPALIVE is OFF

$ LD_PRELOAD=libkeepalive.so \
> KEEPCNT=20 \
> KEEPIDLE=180 \
> KEEPINTVL=60 \
> test
SO_KEEPALIVE is ON
TCP_KEEPCNT   = 20
TCP_KEEPIDLE  = 180
TCP_KEEPINTVL = 60
```

Et vous pouvez utiliser **strace** (1) pour comprendre ce qui se passe:

```
$ strace test
execve("test", ["test"], [/* 26 vars */]) = 0
[..]
open("/lib/libc.so.6", O_RDONLY)        = 3
[..]
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
getsockopt(3, SOL_SOCKET, SO_KEEPALIVE, [0], [4]) = 0
close(3)                                = 0
[..]
_exit(0)                                = ?

$ LD_PRELOAD=libkeepalive.so \
> strace test
execve("test", ["test"], [/* 27 vars */]) = 0
[..]
open("/usr/local/lib/libkeepalive.so", O_RDONLY) = 3
[..]
open("/lib/libc.so.6", O_RDONLY)        = 3
[..]
open("/lib/libdl.so.2", O_RDONLY)       = 3
[..]
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_SOCKET, SO_KEEPALIVE, [1], 4) = 0
setsockopt(3, SOL_TCP, TCP_KEEPCNT, [20], 4) = 0
setsockopt(3, SOL_TCP, TCP_KEEPIDLE, [180], 4) = 0
setsockopt(3, SOL_TCP, TCP_KEEPINTVL, [60], 4) = 0
[..]
getsockopt(3, SOL_SOCKET, SO_KEEPALIVE, [1], [4]) = 0
[..]
getsockopt(3, SOL_TCP, TCP_KEEPCNT, [20], [4]) = 0
[..]
getsockopt(3, SOL_TCP, TCP_KEEPIDLE, [180], [4]) = 0
[..]
getsockopt(3, SOL_TCP, TCP_KEEPINTVL, [60], [4]) = 0
[..]
close(3)                                = 0
[..]
_exit(0)                                = ?
```

Pour d'autres informations, visitez la page d'accueil du projet libkeepalive : http://libkeepalive.sourceforge.net/