Rapport PFE

Composant de supervision des interfaces réseaux d'un OS

Evry, le 29/01/2013
Télécom et Management SudParis

- **Encadré par:** M. Sébastien LERICHE
- **Réalisé par :**
  - Al Mostafa SAHIM
  - Amine KARIM

Remerciements

On tient tout d'abord à remercier profondément tous ceux qui ont contribué à la bonne marche et la réussite de notre semestre d'échange à Télécom SudParis, pour tous les efforts qu'ils n'ont cessés de fournir, pour mener à bien nos études.

On tient également à exprimer notre profonde gratitude et nos remerciements les plus sincères principalement à tout le staff ASR de Télécom SudParis qui nous ont offert l'opportunité d'évoluer au sein de leur entité, et spécialement pour M. Michel SIMATIC & notre encadrant M. Sébastien LERICHE pour leur prestigieux encadrement et leur précieuse assistance.

Sommaire

Liste des figures

Introduction

Le laboratoire d'informatique de l'ENAC (Ecole Nationale de l'Aviation Civile) s'intéresse à une nouvelle manière de concevoir des programmes interactifs (projet ANR ISTAR ou I*). Dans leur proposition, chaque périphérique d'un système d'exploitation peut être vu comme un composant logiciel, capable d'émettre ou de recevoir des évènements. Par la suite, la conception d'une application consiste à décrire les interactions qui pourront se dérouler entre les composants, par le biais des évènements.

Le sujet de notre projet consiste à réaliser un composant utilisable dans ISTAR qui puisse fournir des évènements sur les changements au niveau des interfaces réseau d'un système d'exploitation, en l'occurrence Linux.

Ledit composant, développé en langage c, devrait réagir directement aux changements qui se produisent au niveau réseau, en particulier lorsqu'on branche/débranche un câble réseau, si les adresses IP ou MAC changent et enfin lorsqu'il ya une stagnation du débit au niveau d'une interface réseau.

Nous allons présenter dans un premier temps les solutions retenues pour accéder aux informations à partir de l'espace utilisateur de Linux, ensuite nous allons aborder le mécanisme de gestions des événements basé sur les signaux. Nous présenterons par la suite l'implémentation de différentes fonctionnalités du composant ainsi que quelques prises d'écran de l'exécution.

I. Cadre du projet :

Le sujet de ce stage consiste à réaliser un composant utilisable dans ISTAR qui puisse fournir des évènements sur les changements au niveau des interfaces réseau d'un système d'exploitation.

Le composant doit permettre de :

Superviser l'état des interfaces réseaux : Le composant doit remonter un événement si un câble est branché/débranché, si l'interface wifi est activée/désactivée …

Superviser les adresses IP des interfaces : Le composant doit déclencher un événement à la suite du changement de l'adresse IP d'une interface connectée.

Superviser les adresses MAC des interfaces : Le composant doit signaler le changement de l'adresse MAC d'une interface.

Superviser la stagnation du débit des interfaces : Le composant doit déclencher un événement à la suite de la stagnation du débit (nombre de paquets transitant) au bout d'une durée déterminée (10 secondes).

Le composant est développé en langage C et notre choix de l'OS a porté sur Linux.

II. Accès aux informations réseaux sous Linux :

Afin de comprendre les contraintes d'accès aux informations réseaux sous Linux, on propose de faire le point sur l'architecture Linux.

1. Architecture de Linux :

L'architecture Linux repose en gros sur deux espaces : L'espace utilisateur et l'espace Kernel ou noyau. Ce dernier gère la couche la plus interne qu'est le hardware. La particularité de ce système est que l'utilisateur n'est pas en mesure d'interagir directement avec le noyau pour gérer la couche matérielle et plus particulièrement les aspects réseaux.

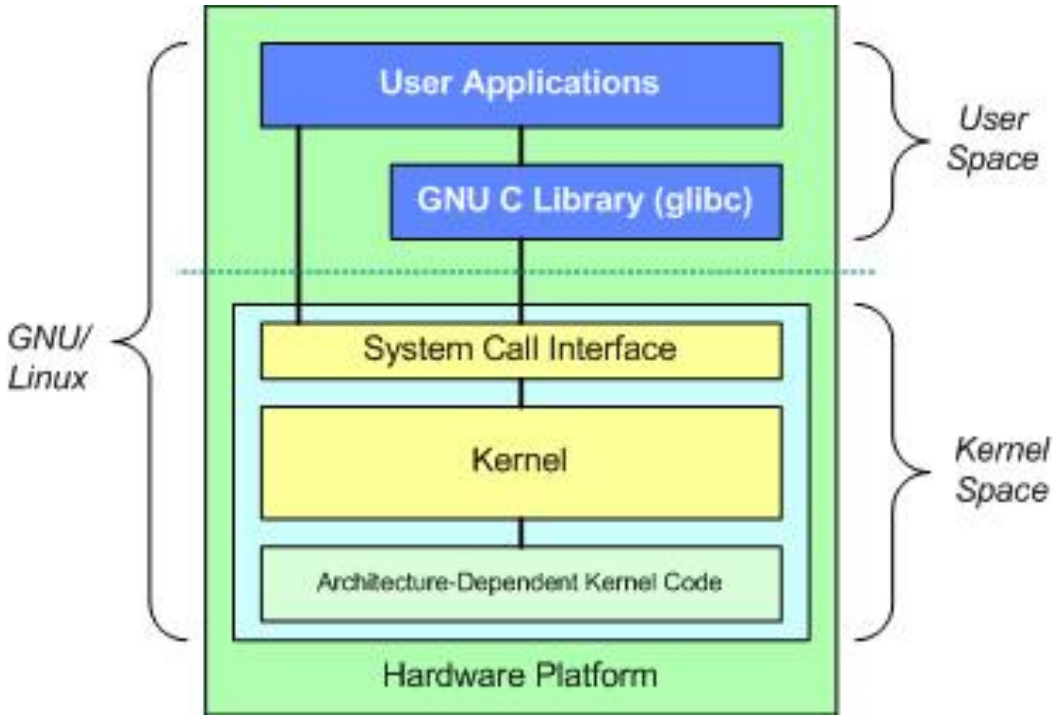

Figure n°1 : Architecture de Linux (1)

Par conséquent , l'accès aux informations des interfaces réseaux sera fait par l'intermédiaire des appels systèmes fournie par l'interface « System Call Interface » ainsi que les pseudo-fichier.

1 :http://www.ibm.com/developerworks/linux/library/l-linux-kernel/figure2.jpg

2. Les appels Système :

Un appel système est un moyen de communiquer directement avec le noyau de la machine. Le noyau regroupe toutes les opérations vitales de la machine. Ainsi il est impossible d'écrire directement sur le disque dur. L'utilisateur doit passer par des appels systèmes qui contrôlent les actions qu'il fait. Ceci permet de garantir :

- la sécurité des données car le noyau interdit à un utilisateur d'ouvrir les fichiers auxquels il n'a pas accès.
- l'intégrité des données sur le disque. Un utilisateur ne peut pas par mégarde effacer un secteur du disque ou modifier son contenu.
- Les fonctions sont contenues quant à elles dans des bibliothèques. Il s'agit donc de code s'exécutant dans l'espace utilisateur. Les fonctions peuvent utiliser des appels système.

Ceci dit , nous avons opté pour les appels systèmes pour identifier les interfaces réseaux connectés , leurs adresses physiques et logiques .ceci est possible grâce aux appels systèmes ioctl

2.1 IOCTL :

Ioctl, raccourci signifiant input-output control est un appel système pour des opérations d'entrée/sortie spécifiques à un périphérique qui ne peuvent être exécutées par un appel système classique. Il reçoit un paramètre spécifiant un code-requête à exécuter ; l'effet de cet appel dépend complètement du code-requête. Les codes-requête sont souvent spécifiques au périphérique.

Les O.S. de type Unix (Linux, FreeBSD, MacOS X) utilisent les ioctls pour configurer les interfaces réseau. On peut par exemple configurer le masque de sous-réseau en ouvrant une Socket puis en utilisant l'ioctl "SIOCSIFNETMASK" sur celle-ci. Il est à noter que la commande ifconfig utilise ioctl pour récupérer les informations (adresse IP , flags , adresse MAC…) qu'elle affiche, on peut facilement vérifier ces appels par la commande

$ strace ifconfig

Linux supporte des ioctls standard pour configurer les périphériques réseau qu'on décrira ci-dessous.

2.1.1 Prototype de la fonction ioctl coté utilisateur :

La fonction ioctl, côté utilisateur a ce prototype:

int ioctl(int *fd*, int *cmd*, char **argp*);

où : *fd* = Le descipteur de fichier dans le réseau.
cmd = (voir 2.1.3 Les commandes de ioctl)
argp= (voir 2.1.2 Les arguments de ioctl)

2.1.2 Les arguments de ioctl :

Pour récupérer les informations des périphériques ou pour les configurer , les ioctl utilisent une structure de données particulière nommée ifreq . Ladite structure est définie comme suit :

```
struct ifreq {
    char ifr_name[IFNAMSIZ]; /* nom interface */
    union {
        struct sockaddr ifr_addr;
        struct sockaddr ifr_dstaddr;
        struct sockaddr ifr_broadaddr;
        struct sockaddr ifr_netmask;
        struct sockaddr ifr_hwaddr;
        short           ifr_flags;
        int             ifr_ifindex;
        int             ifr_metric;
        int             ifr_mtu;
        struct ifmap    ifr_map;
        char            ifr_slave[IFNAMSIZ];
        char            ifr_newname[IFNAMSIZ];
        char           *ifr_data;
    };
};
```

On retiendra particulièrement de cette structure de données les champs qui vont nous servir dans notre composant de supervision à savoir :

- `ifr_name` : sert à récupérer le nom de l'interface , par exemple « etho » .
- `ifr_hwaddr` : une structure de données contenant l'adresse mac de l'interface.
- `ifr_addr` : une structure de données contenant l'adresse ip de l'interface.

2.1.3 Les commandes de ioctl

Comme indiqué dans le prototype de la fonction ioctl , cette dernière utilise des commandes spécifiques pour récupérer une information sur un périphérique réseaux.Ci-dessous la liste des commandes :

SIOCGIFNAME

En utilisant *ifr_ifindex*, renvoie le nom de l'interface dans *ifr_name*. C'est le seul ioctl qui renvoie un résultant dans *ifr_name*.

SIOCGIFINDEX

Retrouve le numéro d'interface et le place dans *ifr_ifindex*.

SIOCGIFFLAGS, **SIOCSIFFLAGS**

Lire ou écrire les attributs actifs du périphérique. *ifr_flags* est un mot contenant un masque de bits combinant les valeurs des flags existants.[1]

SIOCGIFMETRIC, **SIOCSIFMETRIC**

Lire ou écrire la métrique du périphérique en utilisant *ifr_metric*. Ceci n'est pas encore implémenté, il renvoie dans *ifr_metric* la valeur 0 si on essaye de lire, et renvoie l'erreur **EOPNOTSUPP** si on essaye d'écrire.

SIOCGIFMTU, **SIOCSIFMTU**

Lire ou écrire le MTU (Maximum Transfer Unit) du périphérique avec *ifr_mtu*. Fixer le MTU est une opération privilégiée. Fixer un MTU trop petit peut faire planter le noyau.

SIOCGIFHWADDR, **SIOCSIFHWADDR**

Lire ou écrire l'adresse matérielle du périphérique en utilisant *ifr_hwaddr*. Cette adresse matérielle est indiquée dans une structure *sockaddr*. *sa_family* contient le type de périphérique ARPHRD_*, *sa_data* est l'adresse matérielle L2 commençant par l'octet 0. Écrire l'adresse matérielle est une opération privilégiée.

SIOCSIFHWBROADCAST

Fixer l'adresse de broadcast du périphérique à partir de *ifr_hwaddr*. C'est une opération privilégiée.

SIOCADDMULTI, **SIOCDELMULTI**

Ajouter ou supprimer une adresse des filtres multicast du niveau liaison du périphérique en utilisant *ifr_hwaddr*. Ce sont des opérations privilégiées.

SIOCGIFTXQLEN, **SIOCSIFTXQLEN**

Lire ou écrire la taille de la file d'émission du périphérique avec *ifr_qlen*. L'écriture de la taille de la file est une opération privilégiée.

SIOCSIFNAME

Changer le nom de l'interface indiquée dans i*fr_name* pour *ifr_newname*. C'est une opération privilégiée. Elle n'est possible que si l'interface n'est pas en fonctionnement.

SIOCGIFCONF

Renvoie une liste des adresses (couche de transport) des interfaces. Ceci ne fonctionne actuellement qu'avec les adresses **AF_INET** (IPv4) pour des raisons de compatibilité. L'utilisateur passe une structure *ifconf* en argument à l'ioctl. Elle contient un pointeur sur une table de structures *ifreq* dans son membre *ifc_req* et la longueur en octets dans *ifc_len*. Le noyau remplit les ifreqs avec toutes les adresses L3 des interfaces en fonctionnement : *ifr_name* contient le nom de l'interface (eth0:1 etc.), et *ifr_addr* l'adresse.

Dans notre composant, on a retenu les ioctls SIOCGIFCONF pou récupérer la liste des adresses IP, SIOCGIFHWADDR pour récupérer les adresses MAC , et SIOCGIFNAME pour récupérer les noms des interfaces connectés.

3. Les pseudo fichiers Linux :

La cahier de charge du projet stipule qu'il faudra superviser la stagnation du débit au niveau d'une interface, les informations concernant le nombre de paquets transitant par l'interface n'étant pas fournies par les appels ioctl. La solution est bien des pseudos fichiers Linux à savoir **/proc/net/dev** ou bien encore **/sys/class/net/x/statistics** (x désignant le nom de l'interface).

3.1 /proc/net/dev :

Ce pseudo-fichier contient des informations d'état sur les périphériques réseau. On y trouve les nombres de paquets émis et reçus, le nombre d'erreurs et de collisions, ainsi que d'autres données statistiques. Ce fichier est utilisé par le programme ifconfig. Le format est le suivant :

```
$ cat /proc/net/dev
Inter-|   Receive                                                |  Transmit
 face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs drop fifo colls carrier compressed
    lo: 247679812  136616    0    0    0     0          0         0 247679812  136616    0    0    0     0       0          0
  em1: 2309187862 5424017    0    0    0     0          0   1143970 3452795494 5761987    0    0    0     0       0          0
$
```

Figure n°2 : contenu du pseudo-fichier /proc/net/dev

Ce fichier demeure une bonne solution pour la supervision de la stagnation du débit au niveau des interfaces réseaux, puisqu'il est possible de relever le nombre de paquets émis et reçus par l'interface .l'inconvénient de ce fichier réside dans la difficulté de le « parser ».

3.2 Le répertoire /sys/class/net/x/statistics :

Ce répertoire Linux contient plusieurs pseudo-fichiers représentant les statistiques d'une interface réseaux donnée. La figure ci-dessous décrit le contenu du répertoire d'une interface connectée.

```
$ ls  /sys/class/net/em1/statistics
collisions  rx_compressed  rx_errors       rx_length_errors  rx_packets        tx_carrier_errors  tx_errors          tx_packets
multicast   rx_crc_errors  rx_fifo_errors  rx_missed_errors  tx_aborted_errors  tx_compressed      tx_fifo_errors     tx_window_errors
rx_bytes    rx_dropped     rx_frame_errors  rx_over_errors    tx_bytes          tx_dropped         tx_heartbeat_errors
$
```

Figure n°3 : Contenu du répertoire /sys/class/net/x/statistics

Si on veut récupérer le nombre de paquets reçus par l'interface, il suffira de lire la valeur qui existe sur le pseudo fichier rx_packets (Figure 4). D'ailleurs, tous les fichiers de ce répertoire contiennent une seule valeur, contrairement au pseudo fichier /proc/net/dev qui contient toutes les informations à la fois, ce qui rend plus facile le fait de récupérer une valeur précise.

```
$ cat /sys/class/net/em1/statistics/rx_packets
5448890
$
```

Figure n°4 : Contenu du fichier /sys/class/net/em1/statistics/rx_packets

L'inconvénient majeur de cette solution est le nombre de fichiers en jeu, en effet pour chaque interface il faudra connaître le nombre de paquets émis et reçus, ce qui fait que pour chaque interface on ouvrera deux fichiers. Cependant, cette méthode présente une facilité remarquable de programmation. Nous avons retenu cette deuxième méthode pour la supervision de stagnation du débit au niveau d'une interface.

III. Mécanisme de gestion des événements :

Un signal est un message envoyé à un processus pour indiquer l'occurrence d'un événement. Le corps du message est en fait très simple puisqu'il est constitué en tout et pour tout d'un entier indiquant le type du signal. Typiquement, l'arrivée d'un signal interrompt, plus ou moins brutalement l'exécution du processus qui le reçoit. Toutefois, on peut redéfinir la procédure de réponse par défaut en installant un gestionnaire, autrement dit une fonction qui va être exécuté à la suite de la réception d'un signal.

Dans notre cas, un signal doit être envoyé à la suite du branchement/débranchement d'un câble, du changement des adresses IP ou MAC d'une interface ou de la stagnation du débit au niveau d'une interface. Nous allons décrire les différentes possibilités d'émission de signal, ainsi que la solution retenue.

1. Emission des signaux :

Un signal peut être émis par :

- **Le noyau** : à la suite d'une division par zéro, à la fin d'un processus (émission du signal SIGHLD),...
- **Un utilisateur** : En utilisant le clavier (<ctrl c>..) ou la commande *kill* du shell
- **Un processus** : appel à la fonction *kill*, à la fonction *alarm*.

Dans notre cas, le signal sera envoyé par le processus. En parcourant les possibilités de l'envoi d'un signal, la méthode la plus approprié est l'utilisation de la fonction alarm. Le prototype de la fonction alarm est décrit ci-dessous :

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
```

Exemple :

m(2) :	**Le processus courant recevra le signal SIGALRM dans deux secondes.**

Reste maintenant à définir les méthodes de réception du signal (SIGALRM dans notre cas)et comment mettre en place un gestionnaire de signaux évitant ainsi l'arrêt brusque du programme.

2. Traitement des signaux reçus :

Suite à la réception d'un signal, on distingue trois options de traitement de ce dernier.

- **Ignorer le signal** : Typiquement en utilisant la fonction avec le prototype *signal (Num_Sig, SIG_IGN)*, ou Num_Sig représente le numéro du signal et SIG_IGN représente une constante indiquant que le signal doit être ignoré.

- **Reprendre le traitement par défaut** : En utilisant la fonction avec le prototype *signal (Num_Sig, SIG_DFL)*, ou SIG_DFL est une constante indiquant que le signal doit être ignoré.

- **Adopter un traitement spécifique :** En utilisant la fonction avec le prototype *signal (Num_Sig, fonction*), ou fonction représente ce qui doit être exécuté à la suite de la réception d'un signal.

Dans notre cas,nous avons retenu la dernière méthode, qui consiste à exécuter une fonction. Cette dernière affiche l'événement survenu au niveau de l'interface réseau signalé par un SIGALRM. Les fonctions restent ouvertes à des modifications de leurs comportements suivant les besoins des utilisateurs de ISTAR.

IV. Implémentation du composant :

Une fois la phase de choix des solutions d'accès aux informations sur les interfaces réseaux ainsi que le mécanisme de gestion d'événements achevé, on s'est mis à coder et à tester les différents cas de figures, possibles bien sur de la réaction du composant aux changements niveau réseau.

La hiérarchie du projet est décrite comme suit :

```
`-- projet
    |-- check_console.h
    |-- main.c
    |-- Makefile
    |-- superviser.c
    `-- superviser.h
```

1. Le fichier superviser.h :

Ce fichier comporte les prototypes des fonctions de supervision et de réponse à un événement survenu au niveau réseau. Ci-dessous le contenu de façon très abstraite de ce fichier :

```
void handler_up(int Num );
void handler_down(int Num );
void handler_ip(int Num );
void handler_mac(int Num );
void handler_debit_loopback(int Num );
void handler_debit(int Num );


void  *superviser();
void  *superviser_mac();
void  *superviser_ip();
void  *superviser_debit();
```

Figure n°5 : prototypes de fonctions contenus dans le fichier superviser.h

Les fonctions handler_x(int Num) sont des fonctions exécutés suite à la récéption du signal SIGALRM par une fonction de supervision. Ces fonctions

se contentent dans un premier temps d'afficher l'événement survenu au niveau de l'interface.

- Void handler_up(int Num) : cette fonction est appelée suite au branchement d'un cable réseau ou l'activation d'une nouvelle interface.
- Void handler_down(int Num) : cette fonction est exécuté lorsqu'un cable est débranché ou un interface a été désactivée.
- Void handler_ip(int Num) : cette fonction est appelée suite au changement de l'adresse IP au niveau d'une interface.
- Void handler_mac(int Num) : cette fonction est exécuté suite au changement de l'adresse MAC d'une interface réseau.
- Void hanlder_debit_loopback(int Num) : cette fonction est appelée suite à la stagnation du débit au niveau de l'interface LoopBack , on a distingué car cette interface ne transitent pas beaucoup de paquets . Un débit inchangé sur cette interface n'est donc pas méchant.
- Void handler_debit(int_Num) : cette fonction est executé suite à la stagnation de débit au niveau d'une interface autre que celle de la boucle locale.Il est à noter qu'après concertation avec notre encadrant , un débit stagne est synonyme au fait qu'aucun paquets émis et réçus n'a transité par l'interface durant 10 secondes.

Les fonctions : **supervision()**, **supervision_ip()** et **supervision_mac()** ont le même principe, elles utilisent des appels systèmes ioctl. Une fois récupérées, le programme stocke les informations dans une liste chainée, temporise, puis compare la liste avec la dernière liste construite dans l'itération i-1. S'il y a un changement dans les deux listes, le processus exécute une fonction alarm(1). Cette fonction, émet un signal SIGALRM au bout d'une seconde et elle est captée par l'une des fonctions handlers évoqués ce-dessus.

La fonction **superviser_debit()** présente un autre concept, vu que le nombre de paquets émis et réçus n'est pas directement récupérables par les appels système ioctl. Le processus lit, pour chaque interface connectée, les valeurs du nombre de paquets transitant par l'interface et les stockent dans un tableau. Le processus temporise 10 secondes, prend une autre fois les valeurs et les compare à ceux de l'itération i-1. S'il ya du changement, le processus émet un signal SIGALRM comme pour les autrs cas, ce signal est capté par le handler correspondant.

2. Fichier check_console.h :

Ce fichier contient une méthode qui lit un caractère de la console, il sera passé en paramètre à un Thread. L'intérêt étant d'écouter sur la console la saisie d'un nombre qui va arrêter tous les threads de supervision lancés.

3. Fichier main.c :

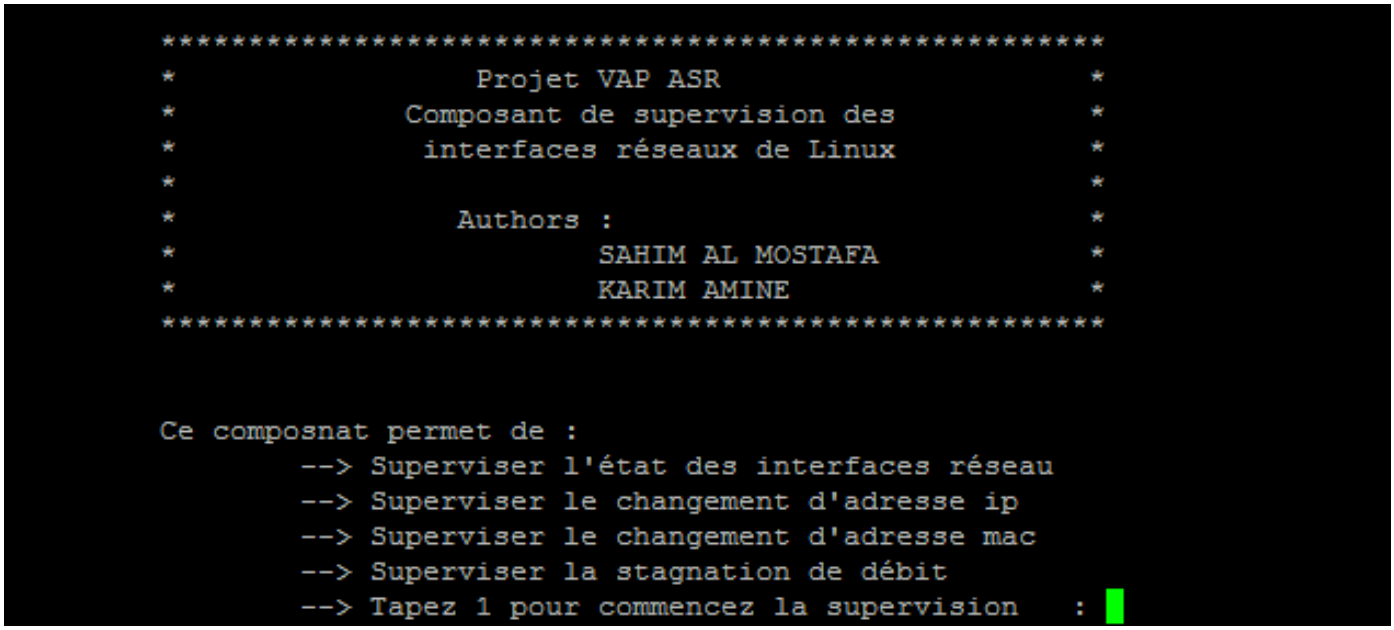

Figure n°6 : Prise d'écran de la fenêtre d'accueil

Au lancement du composant un écran d'accueil apparaît, l'utilisateur doit saisir le nombre 1 pour lancer la supervision. Lancer la supervision correspond à la création de 5 threads comme décrit ci-dessous :

```
retour= pthread_create(&thread1, NULL, check_console, NULL);
retour= pthread_create(&thread2, NULL, superviser, NULL);
retour= pthread_create(&thread3, NULL, superviser_ip, NULL);
retour= pthread_create(&thread4, NULL, superviser_mac, NULL);
retour= pthread_create(&thread5, NULL, superviser_debit, NULL);
```

Cette séquence du code lance 4 threads avec en paramètre les quatre fonctions de supervision à savoir la supervision de l'état des interfaces, la supervision des adresses IP, la supervision des adresses MAC et enfin la supervision du débit. Le premier thread lancé boucle sur la console et attend la saisie d'un nombre à tout moment pour arrêter comme il faut la supervision.

Une fois la supervision lancée, le programme affiche en permanence des messages sur la console. Si un message est affiché en rouge, c'est qu'un événement a survenu. Par exemple : la figure 7 décrit un événement (le mécanisme de signaux/ gestionnaire de signaux a fonctionné), celui de la stagnation du débit au niveau de l'interface de LoopBack.

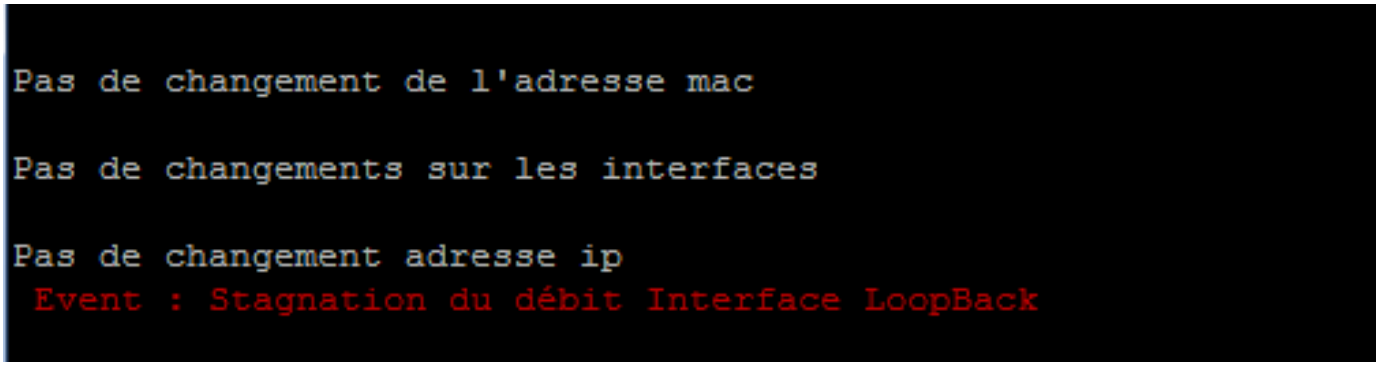

Figure n°7 : Prise d'écran de l'exécution du composant

Conclusion

Finalement ce projet nous a été d'une grande importance et nous a parmi de découvrir des nouvelles notions surtout concernant les méthodes de récupération des informations concerant les interfaces réseaux à partir de noyeau du système Linux et leurs exploitation sous forme d'un composant capable de réagir via des évenements avec les autres composants du réseau.

Les objectifs atteints

La création d'un composant de supervision des interfaces réseaux de Linux qui permet de détecter les changement dans l'état des interfaces, leurs adresses IP, leurs adresses MAC et enfin leurs débits.

Le travail restant

- L'intégration du composant dans le projet ISTAR
- Développement de composant pour d'autres OS

Bibliographie :

http://www.infres.enst.fr/~dupouy/pdf/BCI/3-ESetSignauxBCI-07.pdf

http://brunogarcia.chez.com/Unix/Docs/Signaux.html

http://linux.die.net/man/7/netdevice

http://lii-enac.fr/en/architecture/istar/docs/libs-cookbook/libs-cookbook.html

Annexe

[1] La liste des flags récupérables

Device flags

IFF_UP	Interface fonctionne.
IFF_BROADCAST	Adresse de broadcast valide.
IFF_DEBUG	Attribut interne de débogage.
IFF_LOOPBACK	Interface de type loopback.
IFF_POINTOPOINT	Interface de type point-à-point.
IFF_RUNNING	Resources allouées.
IFF_NOARP	Pas de protocole Arp, adresse de destination L2 absente.
IFF_PROMISC	Interface en mode promiscuous.
IFF_NOTRAILERS	N'utilise pas les postambules.
IFF_ALLMULTI	Accepte tous les paquets multicast.
IFF_MASTER	Maître d'un système de répartition de charge.
IFF_SLAVE	Esclave d'un système de répartition de charge.
IFF_MULTICAST	Support multicast.
IFF_PORTSEL	Capable de sélectionner le média via ifmap.
IFF_AUTOMEDIA	Sélection automatique du média.
IFF_DYNAMIC	Adresse perdue quand l'interface est arrêtée.