Processus d'ajout d'un nouvel appel système au noyau Linux et son utilisation dans un programme utilisateur

Objectif du laboratoire

- L'objectif de ce laboratoire est de vous familiariser à travailler à l'intérieur du noyau de Linux. En utilisant votre machine virtuelle, vous allez :
- Vous familiariser avec le processus de configuration, compilation, et installation du noyau du système d'exploitation Linux
- Modifier le noyau afin d'ajouter un nouvel appel système et montrer que vous pouvez utiliser cet appel système dans un programme utilisateur simple
- Ajouter un autre appel système plus significatif qui retourne des informations utiles sur le processus en cours d'exécution.

Première partie : Processus de configuration, compilation et installation du noyau Linux

La compilation du noyau et son installation sont des opérations critiques, car toute erreur risque de rendre le système non disponible. Pour cela, vous allez travailler sur une machine virtuelle en VirtualBox sur laquelle est déployée une distribution Ubuntu.
Note : pour avoir le privilège du **root** avec le compte d'utilisateur ordinaire, vous devez utiliser la commande **sudo,** ou bien basculer au compte root par la commande **su -**

Préparation de l'environnement

- Télécharger l'outil VirtualBox et la distribution de Linux comme indiqué dans la section laboratoires de site du cours LOG710.
  - Pour savoir la version du kernel actuel : **uname -r**
- Télécharger le code source du Kernel

Voici l'une des façons : **sudo apt-get install linux-source**
Peut-être on aura besoin de : **sudo apt-get update** et d'autres paquets comme **libncurses5-dev**

- Décompresser le fichier du code source linuxsource-3.13.0.tar.bz2 (/usr/src) au niveau de votre répertoire home **/home/user** en utilisant la commande **tar jxvf** (où user est le non de l'utilisateur du system (ex. vagrant)). Cette opération va créer le répertoire /home/user/linux-source-3.13.0 contenant toute l'arborescence du code source du noyau.
  À titre d'exemple nous avons utilisé la désignation 3.13.0 qui dépend de la version du noyau utilisé ($(uname -r)).

Processus de compilation et installation
Le processus de compilation du noyau Linux est relativement simple et standardisé.
Ce processus comporte essentiellement trois étapes :
1. Configuration du noyau
2. Compilation du noyau et des modules du noyau
3. Installation des modules et du noyau
Les trois étapes se font avec la même commande **make** au niveau du répertoire racine de l'arborescence du code source du noyau mais avec des options différentes. Pour avoir de l'information sur toutes les possibilités vous pouvez utiliser la commande **man make**.

Étape de configuration du noyau :
Cette étape est nécessaire avant de compiler le code du noyau. Il est possible de compiler le noyau avec un support pour des caractéristiques (features) particulières et/ou un matériel spécifique. La configuration du noyau est contrôlée par des options de configuration de la forme CONFIG_FEATURE. Par exemple, le support du parallélisme symétrique (symmetrical multiprocessing (SMP)) dans le noyau est contrôlé par l'option de configuration CONFIG_SMP. Quand cette option est activée, le SMP est activé sinon il est désactivé. Les options de configurations sont utilisées pour spécifier les fichiers sources du noyau à compiler.
Si on utilise la même version du système ou deux versions proches, il est recommandé de réutiliser la même configuration fonctionnelle (installée) et l'optimiser ensuite si nécessaire.
Pour cela, il faut copier le fichier de configuration **config-3.13.0-74-generic** (existant dans /**boot**) dans le répertoire racine contenant le code source du noyau à compiler : **/home/user/linux-source-3.13.0**.
Une fois le fichier **.config** copié, il faut utiliser la commande suivante pour le prendre en considération dans la configuration du nouveau noyau : **make oldconfig.**

Si les deux versions sont trop différentes, **make oldconfig** sera difficile à utiliser.
Il est recommandé alors de construire une configuration minimum du noyau par la commande **make defconfig** qui construit un .config minimum fourni par les responsables du noyau. Puis à remplir les options dont vous aurez besoin à travers l'outil de configuration (**make menuconfig**).

Le seul changement à faire pour le besoin de ce laboratoire vise à bien identifier le noyau que vous compilez. Pour cela, il faut initialiser la variable EXTRAVERSION du fichier Makefile avec la chaine :
-log710a2018GXXYYZZ où XX, YY et ZZ sont les initiales des membres de votre équipe.
Vous pouvez éditer le fichier MakeFile en utilisant un éditeur de fichiers texte comme *nano*.

Étape de compilation du noyau et des modules :

Cette étape consiste à compiler le code du noyau (avec **make**) et les modules du noyau (avec **make modules**) en tenant compte de la configuration établie dans l'étape précédente.
Pour accélérer la compilation, il faut exécuter la commande **make** au niveau du répertoire racine de l'arborescence du code source du noyau avec l'option **–j** suivi par un nombre qui représente le nombre de threads (il est recommandé qu'il soit égal au nombre de cœur de la CPU + 1). Exemple : **make modules –j3**
Note : Dans ce laboratoire nous allons suivre la méthode universelle de la compilation. Puisqu'il y a d'autres manières de compilation spécifiques aux autres distributions.

Étape d'installation du noyau et des modules :
Pour installer le noyau et les modules qui viennent d'être compilés, il faut utiliser la commande **make** avec les options **modules_install** et **install** respectivement. Ceci crée les fichiers contenant le noyau au niveau du répertoire **/boot** et les fichiers des modules du noyau dans un répertoire spécifique à la version de votre noyau au niveau du répertoire **/lib/modules**.

La commande (make install) crée automatiquement un fichier **initrd.img-3.13.0-log710a2018GXXYYZZ** qui est nécessaire pour démarrer la machine avec le nouveau noyau compilé. Elle permet également de mettre à jour le boot loader (grub).

Redémarrage de la machine et vérification
Rebooter votre machine et choisir dans le menu du boot loader votre version du noyau. Si le système vous demande de vous logger, c'est que l'opération est réussie. Pour vérifie que c'est bien votre noyau qui tourne dans votre machine, utiliser la commande **uname –r.**

Note : Si nous voulons faire une autre compilation du noyau, il faut utiliser la commande « **make clean** » et « **make mrproper**» (supprime aussi le .config) pour effacer les fichiers générés (fichiers binaires) tout en gardant le code source et la configuration.

Deuxième partie : Processus d'ajout et utilisation d'un premier appel système

Dans cette partie de ce laboratoire, vous allez modifier légèrement le noyau Linux de votre machine virtuelle pour ajouter un nouvel appel système.

Le premier appel système que vous allez ajouter dans cette partie ***sys_log710a2018as1*** va simplement afficher un message dans le journal du système (system log) pour indiquer qu'il a été appelé et retourne immédiatement. Cet appel système utilise la fonction noyau **printk** qui est l'équivalent de la fonction **printf()** de la librairie C standard.

Dans ce qui suit, pour alléger le texte, on va supposer que le répertoire racine du code source du noyau est **noyauSource** (vous pouvez utiliser la commande **ln** pour créer un raccourci).

Vous avez besoin de référencer votre appel système dans le noyau. Pour cela vous aurez à :

- Modifier la table des appels système (le fichier **syscall_32.tbl** pour 32 bit et **syscall_64.tbl** pour 64 bits) qui se trouve dans le répertoire **noyauSource/arch/x86/syscalls** pour définir le numéro du nouvel appel système. Vous y allez trouver une liste d'autres appels système existants. Votre entrée dans ce fichier doit suivre le même format et l'ajouter à la fin. Le nom de votre appel système est **log710a2018as1.** Alors que le nom de la fonction est **sys_log710a2018as1**(par convention, il commence par **sys**_) et le numéro à utiliser doit être plus grand d'une unité que le dernier appel système existant.

- Modifier le fichier (header) **syscalls.h** qui se trouve dans le répertoire **noyauSource/include/linux** en ajoutant la signature de la fonction de l'appel système (à la fin du document avant #endif).

```
asmlinkage long sys_log710a2018as1(void);
```

Afin de séparer l'implémentation de votre appel système aux autres appels, créer un sous-répertoire (**noyauSource/partie2**).

Saisir l'implémentation de votre appel système sys_log710a2018as1 dans un nouveau fichier **noyauSource/partie2/syscall1_log710.c**

```
#include <linux/kernel.h>
#include <linux/syscalls.h >

asmlinkage long sys_log710a2018as1(void) {
   printk(KERN_ALERT "LOG710 AUTOMNE 2018 Appel Systeme 01!\n");
   return 0;
}
```

Notez qu'il n'y a pas de virgule entre **KERN_ALERT** et le message.

Vous avez besoin de créer un fichier **Makefile** spécifique à votre appel système. Saisir la ligne suivante :

```
obj-y := syscall1_log710.o
```

Elle ajoute l'objet créé à partir du fichier **`syscall1_log710.c`** à la liste d'objets du noyau à compiler. Pour cela, vous devez ajouter le répertoire **partie2/** pour l'attribut **core-y** dans le fichier Makefile du noyau**.**

Il est préférable de procéder la compilation du module de l'appel système avant la compilation du noyau, par la commande **make M=partie2/**. De cette manière, on peut détecter rapidement si le module a une erreur au lieu de la détecter durant la compilation du noyau.

Ceci complète le processus d'ajout du nouvel appel système. **Il faut recompiler et réinstaller le noyau (sans les modules) avec ces modifications comme vous l'avez fait dans la première partie du lab**. Modifier la valeur de l'attribut **EXTRAVERSION** dans Makefile du noyau à **–log710a2018GXXYZZlab02P2** pour distinguer cette version du noyau (XXYYZZ sont les initiales des membres de votre équipe).

Une fois vous avez un nouveau noyau modifié avec ce nouvel appel système vous pouvez écrire un programme utilisateur qui utilise cet appel système afin de le tester. Pour cela, utiliser le programme suivant :

```
#include <sys/syscall.h>
#include <stdio.h>

#define sys_log710a2018as1     999   /* utiliser le numéro d'appel système que vous
                                        avez choisi dans le fichier syscall_32.tbl */

long foncTestAS1 ( void ) {
        return (long) syscall(sys_log710a2018as1);
};

main () {
   printf("Le code de retour du nouvel appel système est : %d\n",
foncTestAS1());
   return 0;
}
```

La fonction **`syscall()`** (voir sa description avec man) invoque l'appel système dont le numéro est donné dans son premier argument. La fonction **foncTestAS1** n'est qu'un wrapper qui fait appel à syscall().

L'exécution de ce programme de test invoque un appel système qui génère un message de test dans un fichier journal circulaire (log) : **/var/log/syslog**. Vous pouvez consulter ce fichier (journal) avec la commande **cat /var/log/ syslog** ou bien en utilisant le programme **dmesg**. Si votre appel système est bien installé, vous devez trouver le message **LOG710 AUTOMNE2018 Appel Systeme 01!** dans ce fichier.

Troisième partie : Appel système retournant des infos sur le processus courant

Dans cette partie du laboratoire, vous allez implémenter un nouvel appel système plus élaboré en suivant le processus décrit dans la deuxième partie. Cet appel système fournit des informations sur le processus courant (celui qui fait cet appel système). Ajouter cet appel système au noyau utilisé dans la deuxième partie.

Cet appel système ***sys_log710a2018as2*** prend comme paramètre un pointeur sur une structure de données ***procdata*** dans l'espace utilisateur. L'appel système va mettre dans cette structure de données les informations sur le processus courant. L'appel système retourne zéro si l'appel est avec succès ou une indication d'erreur (-EFAULT) sinon.

La structure ***procdata*** est définie comme suit :

```
#include <linux/types.h>

#include <linux/pid.h>

struct procdata {
    long state;     // Etat du processus
    pid_t pid;      // PID du processus
    pid_t parent_pid; // PID du processus père
    uid_t uid;      // ID de l'utilisateur du processus
    char comm[16]; // nom du programme (commande) ayant résulté en ce processus
};
```

Vous devez définir la structure **procdata** dans un fichier **procdata.h**.

Vous pouvez consulter l'implémentation des appels systèmes **`getuid`** and **`getpid`** pour vous guider dans l'Implémentation de votre appel système. Ces deux appels système sont définis dans le fichier **noyauSource/kernel/timer.c**

Toutes les informations dont vous avez besoin pour remplir les attributs (champs) de la structure **procdata** se trouvent dans une structure utilisée par le noyau Linux pour manipuler les taches (processus et threads). Cette structure est appelée **task_struct** et est définie dans le fichier **noyauSource/include/linux/sched.h**.

Une macro définie dans le fichier **noyauSource/include/asm-generic/current.h** retourne l'adresse de l'instance de **task_struct** du processus courant. Cette macro est appelée simplement ***current***

Un appel système doit absolument vérifier la validité des arguments passés par un processus utilisateur appelant. En particulier, un appel système doit bien vérifier les pointeurs fournis par un programme de l'espace utilisateur. Le noyau Linux définit pour

cela deux fonctions qui font ces vérifications et aussi performent un transfert d'informations entre l'espace noyau et l'espace utilisateur. Ces deux fonctions sont ***copy_from_user*** et ***copy_to_user***. Elles sont définies dans le fichier **noyauSource/include/asm-generic/uaccess.h**. Dans ce laboratoire, vous avez besoin d'utiliser la deuxième car l'appel système que vous implémentez transfère des informations sur le processus courant du noyau vers l'espace utilisateur.

Exemple : Supposons qu'au niveau du code d'un appel système vous avez assemblé des données dans une structure de données **kdata** que vous voulez transférer dans une structure pointée par un pointeur **pudata** équivalent dans l'espace utilisateur. Le code correspondant utilisant ***copy_to_user*** serait :

```
/* copier les données de kdata vers une zone dans l'espace
utilisateur pointe par pudata */
if (copy_to_user(pudata, &kdata, sizeof kdata)
          return –EFAULT;
```

EFAULT est un code d'erreur définit dans le fichier **noyauSource/include/asm/errno.h**.

Implémenter votre nouvel appel système ***sys_log710a2018as2*** dans un fichier ***procdata.c***. Il faut modifier le fichier **noyauSource/Makefile** pour ajouter **procdata.o** à la liste des fichiers objets. Ceci permet de compiler et de lier ce fichier au reste du noyau.

Concevoir et implémenter un programme de test dans l'espace utilisateur qui invoque le nouvel appel système. Ce programme doit utiliser **procdata.h** et doit définir une fonction wrapper **getprocdata**(…) qui invoque l'appel système. Ce programme de test doit imprimer toutes les informations retournées par l'appel système. En particulier, tester le cas où le pointeur fournit à l'appel système est NULL ou invalide.

Évaluation du laboratoire

Ce laboratoire doit se faire en équipe et il vaut **10 points** répartis comme suit : Partie 1 (2 points), partie 2 (4 points) et partie 3 (4 points).

Soumission :

- Soumettre un rapport Log710Automne2018Lab02P1XXYYZZ où XX, YY et ZZ sont les initiales des membres de votre équipe.
  Ce document doit inclure pour :
  - **Partie 1** : exactement toutes commandes que vous avez utilisées pour accomplir le processus de compilation et installation du noyau avec une capture d'écran du résultat (le nom de la nouvelle version du noyau).
  - **Partie 2 et 3** : description des étapes suivies avec des captures écrans.

- En plus, il faut soumettre les fichiers des programmes **procdata.c** et le programme utilisateur de test.