TP4 - ASR2 - ENS Lyon - L3IF - 2019-2020
Implémentation d'un syscall

Intervenants:

Stella Bitchebe (celestine-stella.ndonga-bitchebe@ens-lyon.fr)
Lavoisier Wapet (patrick-lavoisier.wapet@ens-lyon.fr)
Alain Tchana (alain.tchana@ens-lyon.fr)
(Note: ce sujet a été rédigé par Stella et Lavoisier)

Objectif général

L'objectif du Tp est d'implanter le syscall make_vcpu() qui:

- Prend en paramètre le PID d'un processus
- Marque tous les threads de ce processus comme étant des vCPUs.
- Écrit dans le log du noyau linux les différentes opérations qu'il effectue.

Ensuite il faudra réaliser un programme en user-space qui crée des threads et appelle ce syscall.

Indications relatives à l'implémentation du syscall

Vous devez utiliser la machine virtuelle (VM) créée lors du TP-1 pour recompiler votre noyau linux (en suivant bien sûr les instructions de ce même TP-1) après l'ajout du syscall.

De façon générale, pour implémenter un nouveau syscall dans le noyau linux il faut créer un nouveau dossier pour votre syscall y ajouter votre code (les .c et .h), faire un Makefile pour le compiler et ajouter votre syscall au Makefile global du kernel.

Cependant, il est préférable, lorsque cela est possible, de *réutiliser* au maximum le code ou les fichiers déjà existant dans le noyau linux! Dans votre cas, vous allez rajouter l'implémentation de votre syscall dans un fichier existant (i.e. on ne va pas créer de nouveaux fichiers .h et .c, ni modifier de Makefile). Vous écrirez par exemple l'implémentation (le code) de votre syscall dans le fichier kernel/sched/core.c en utilisant la macro SYSCALL_DEFINE1 qui permet de définir un syscall. Ici le chiffre 1 représente le nombre de paramètres du syscall. Recherchez une utilisation de cette directive dans ce fichier pour vous en inspirer.

Sachez aussi que les threads et les processus sont manipulés par le kernel sous la forme d'une structure de données: *task_struct*, définie dans le fichier */include/linux/sched.h*. Pour marquer un thread (task) comme vCPU, vous ajouterez un champ dans cette structure (qui indiquera par un 1 ou 0 si le processus est un vCPU ou pas). Vous pourrez le nommer par exemple "*is_vcpu*".

Vous aurez besoin de quelques APIs ou fonctions prédéfinies fournies par le noyau pour pouvoir manipuler les processus:

- Rechercher un processus à partir de son pid: *find_process_by_pid*
- Lire et modifier la structure d'un processus de manière sécurisée:

rcu_read_lock, *rcu_read_unlock*
- Parcourir la liste des threads d'un processus: *for_each_thread*

Vous aurez aussi besoin des champs suivants de la structure *task_struct*:

- *pid*: qui est le pid du processus ou du thread (bref d'un task)
- *tgid*: tous les threads d'un processus ont le même tgid qui est égal au pid du task père. Et pour le processus père, *pid = tgid.*

La déclaration d'un nouveau syscall se fait dans les fichiers suivants:

- arch/x86/entry/syscall/syscall_64.tbl: qui est la ***table des syscalls***. Le numéro défini dans ce fichier est "le numéro du syscall", celui qui sera utilisé pour l'invoquer dans une appli c.

```
332 common  statx           sys_statx
333 common  make_vcpu       sys_make_vcpu

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
```

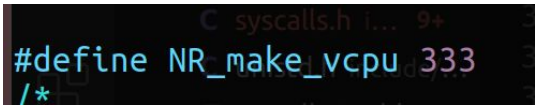
- include/linux/syscall.h: contient la déclaration des fonctions qui implantent les syscalls

```
asmlinkage long sys_statx(int dfd, const char __user *path, unsigned flags,
                          unsigned mask, struct statx __user *buffer);
asmlinkage long sys_make_vcpu(pid_t pid);

#endif
```

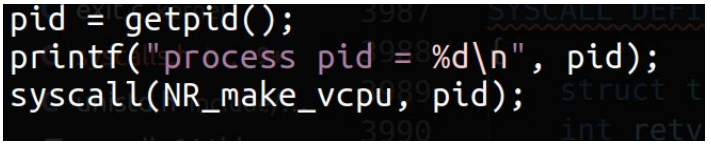
La définition de notre syscall est fait dans core.c:

```
static struct task_struct *find_process_by_pid(pid_t pid)
{
    return pid ? find_task_by_vpid(pid) : current;
}

SYSCALL_DEFINE1(make_vcpu, pid_t, pid)
{
    struct task_struct *p, *thread;
    int retval, count = 1;

    if (pid < 0)
        return -EINVAL;

    retval = -ESRCH;
    rcu_read_lock();
    p = find_process_by_pid(pid);
    if (p) {
        for_each_thread(p, thread)
        {
            if(thread->pid != thread->tgid){
                thread->is_vcpu = 1;
                printk("Thread %d has pid %d and tgid %d and is now a vcpu: thread.is_vcpu = %d\n", count, thread->pid, thread->tgid, thread->is_vcpu);
            }
            count++;
        }

        retval = 0;
    }
    rcu_read_unlock();
    return retval;
}
```

Dans l'application créée pour tester votre syscall, utilisez la fonction *syscall(int number, ...) - vous pouvez regarder le man*.



Ici 333 est le numéro que nous avons défini pour notre syscall dans le fichier syscall_64.tbl



Pour vérifier l'exécution de votre syscall, dans l'implémentation vous pourrez faire des printk (équivalent kernel de printf) dans votre syscall. Et ces prints seront visibles dans les logs du kernel : *sudo dmesg* (ajouter un TAG afin de filtrer, grep, facilement les logs).