Projet : développement d'un mini système d'exploitation pour PC x86

Mise en place des appels systèmes

Jérôme Ermont et Emmanuel Chaput

IRIT - Toulouse INP/ENSEEIHT

Objectifs et mise en place

Comprendre et mettre en œuvre des appels système

- Télécharger le patch sur Moodle
- Patcher le noyau
  - Exécuter `patch -p1 < n7OS_syscall.patch` dans le répertoire n7OS

Un exemple à tester

- L'appel système `example` a été défini dans le patch :

```
int example();
```

  Il retourne la valeur 1 lorsqu'il est exécuté.

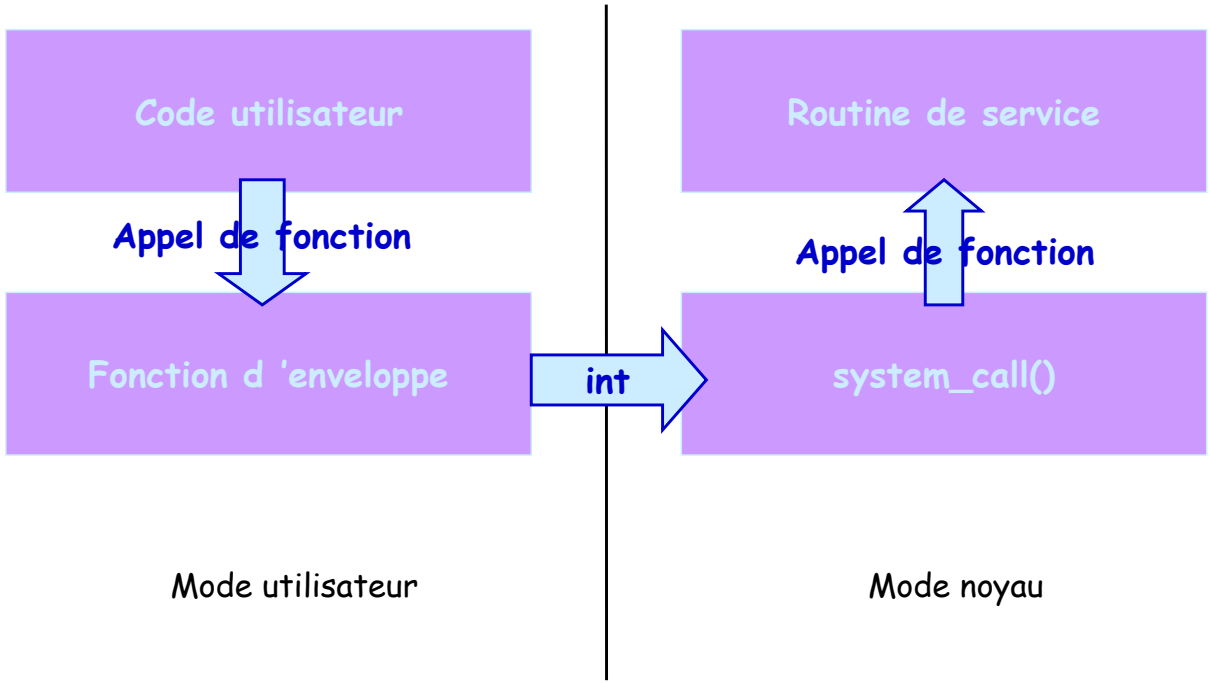
- Pour le tester, dans `start.c` :

```
...
#include <unistd.h>
...
void kernel_start(void) {
    ...
    sti();

    if (example() == 1) {
        printf("Appel␣systeme␣example␣ok\n");
    }
    ...
}
```

Principe des appels système

- Passage incontournable pour accéder aux fonctions du système
- Demande de privilèges



- Passage réalisé par une interruption logicielle pour x86
  - par ex `int 0x80` pour Linux
  - pour rappel une entrée dans l'IDT : champs DPL = niveau de privilège

Écriture de l'appel système `shutdown`

- `shutdown` stoppe l'exécution de qemu suivant la valeur d'entrée `n`.
- Code

```
int sys_shutdown(int n) {
    if (n == 1) {
        outw(0x2000, 0x604); // Poweroff qemu > 2.0
        return -1;
    } else
        return n;
}
```

Étape 1 : ajout de l'appel système à `unistd.h`

- Ajouter le numéro de l'appel système

```
#define NR_shutdown 1
```

- Ajouter l'interface de la fonction

```
int shutdown(int n);
```

Étape 2 : écrire le code de la fonction `shutdown`

- Créer le fichier `shutdown.c` (dans le répertoire lib par exemple)
- Insérer le code :

```
#include <unistd.h>

syscall1(int, shutdown, int, n)
```

- La macro `syscall1` permet d'émettre l'interruption logicielle 0x80. Un paramètre est considéré ici et sera stocké dans le registre `ebx` du processeur.
- 4 types de macros en fonction du nombre de paramètres de la fonction
- Ces macros sont définies dans `unistd.h`

Étape 3 : ajouter l'appel système à la liste

- Modifier le fichier `syscall_defs.h` :

```
#define NB_SYSCALL 2

int sys_shutdown();
```

- Modifier `sys.h` :
  - Ajouter l'appel système à la table `syscall_table` (dans `init_syscall` :

```
void init_syscall(void) {
...
    add_syscall(NR_shutdown, sys_shutdown);
...
}
```

  - Ajouter le code de `sys_shutdown`

Gestionnaire de l'interruption logicielle

```
nr_syscall = 1 # c'est␣le␣nombre␣total␣d'appels systeme
.globl handler_syscall
handler_syscall:
  # %eax contient le numero de l'appel␣systeme
␣␣#␣si␣%eax␣>␣nr_syscall-1␣alors␣erreur
␣␣cmpl␣$nr_syscall-1,␣%eax
␣␣ja␣bad_sys_call
␣␣#␣%ebx␣=␣arg1,␣%ecx␣=␣arg2,␣%edx␣=␣arg3
␣␣#␣on␣empile␣pour␣la␣fonction␣de␣traitement␣de␣l'AS
  pushl %edx
  pushl %ecx
  pushl %ebx
  # appel au code l'AS␣:␣syscall_table[i]␣(i==%eax)
␣␣call␣*syscall_table(,%eax,4)
␣␣#␣on␣re-depile
␣␣popl␣%ebx
␣␣popl␣%ecx
␣␣popl␣%edx
iret
```

- Modifier la variable `nr_syscall` : `nr_syscall = 2`

Travail à faire

- Ajouter l'appel système `write` :

```
// affiche s sur la console
int write(const char *s, int len);
```

- Remplacer `console_putbytes` de `printf` (dans lib) par l'appel à `write`