Exercices de systèmes d'exploitation

Semaine 2: Les appels systèmes et les interruptions

Durant cette séance, nous vous demandons d'écrire sur papier quelques algorithmes en langage d'assemblage illustrant le mécanisme des appels système.

Prélude : appels systèmes Windows en assembleur.

Voici le code assembler d'un appel système en Linux pour afficher « Hello World »

```
;
; This program runs in 32-bit protected mode.
;  build: nasm -f elf -F stabs name.asm
;  link:  ld -o name name.o
;
; In 64-bit protected mode you can use 64-bit registers (e.g. rax instead of
eax, rbx instead of ebx, etc..)
; Also change "-f elf " for "-f elf64" in build command.
;
section .data                           ; section for initialized data
str:     db 'Hello world!', 0Ah         ; message string
str_len: equ $ - str                    ; calcs length of string (bytes

section .text                           ; this is the code section
global _start                           ; _start is equivalent to main() in Java
_start:                                 ; procedure start
        mov    eax, 4                   ; specify the sys_write function code
                                        ; (from OS vector table)
        mov    ebx, 1                   ; specify file descriptor stdout -in
                                        ; linux, everything's treated as a file,
                                        ; even hardware devices
        mov    ecx, str                 ; move start _address_ of string message
                                        ; to ecx register
        mov    edx, str_len             ; move length of message (in bytes)
        int    80h                      ; tell kernel to perform the system call
                                        ; we just set up - in linux services are
                                        ; requested through the kernel
```

Lisez ce code et essayez de le comprendre.
Nous allons dans la suite des exercices écrire ce que l'OS doit faire lors d'un appel système lisant un caractère au clavier.

1. Communication avec les périphériques via le mapping mémoire

Nous allons écrire le code d'une routine système qui lit le prochain caractère disponible au clavier. Cette routine va attendre qu'un caractère soit disponible.
Une fois lue, la valeur du caractère sera mise dans le registre AX et la routine retourne vers le mode utilisateur en utilisant l'instruction IRET.

Nous travaillons sur un ordinateur dans lequel les périphériques sont accédés via un mapping en mémoire.
Le contrôleur de clavier possède deux registres dont les adresses et les rôles sont résumés dans la table suivante:

Adresse	**Registre**	**Valeur**
FFA0h	Contrôle	0 = pas de caractère disponible 1 = un caractère disponible
FFA2h	Caractère	Le code du caractère disponible

2. Communication avec les périphériques via les ports E/S

Réaliser le même programme mais avec l'accès au périphérique via des ports E/S. La table suivante décrit la correspondance entre les ports et les registres du périphérique.

Ports	**Registre**	**Valeur**
01h	Contrôle	0 = pas de caractère disponible 1 = un caractère disponible
02h	Caractère	Le code du caractère disponible

3. Performances

Quel problème de performance voyez à cette façon de lire des caractères?
Comment peut-on la solutionner?

4. Protection mémoire

1) Comme nous l’avons vu durant le cours, pour réellement réaliser l’appel système l’Os utilise une structure de données.
Quelle est-elle ?

2) Où doit-on placer cette structure de données pour éviter qu’un hacker n’arrive à détourner l’OS à son profit ?
Faites un petit dessin avec les limites mémoire de l'OS et la place des composants.