Mécanisme d'interruption

La notion *d'interruption* se base sur un mécanisme par lequel certaines composantes physiques (horloge, E/S, mémoire, processeur) peuvent interrompre le traitement normal du processeur.
Une interruption est la commutation d'un processus à un autre provoquée par un signal du matériel.
Il existe différents types de signaux provoquant une interruption :

- logiciel : division par zéro, référence mémoire en dehors de l'espace autorisé au processus, dépassement de capacité (pile), appel système ;
- temporisateur : le processeur permet à l'OS d'effectuer régulièrement certaines fonctions (ordonnancement, mise à jours) ;
- E/S : signale l'achèvement normal d'une opération ou une erreur ;
- défaillance matérielle : coupure d'alimentation, erreur de parité mémoire.

4.2 Interruptions et appels système du noyau Linux

Un processus peut provoquer une interruption. En assembleur, celle-ci est engendrée par l'instruction `int` suivie du numéro d'un service. Par exemple, pour exécuter une fonction du noyau Linux depuis un code assembleur, on peut utiliser le service `int $0x80`.
Une interruption est alors provoquée i.e. un code constitutif du noyau est exécuté. Pour mener à bien cette exécution, il faut de plus connaître :

- la fonction du noyau à exécuter ;
- comment transmettre les arguments à cette fonction ;
- comment récupérer la sortie de cette fonction.

- Ces informations transitent par certains registres :
- la fonction à exécuter est définie par le contenu du registre %eax et la table 3.1 ;
- les arguments à transmettre sont déterminés par les registres %ebx, %ecx, %edx, %esi et %edi ;
- le code de retour de la fonction est stocké dans le registre %eax. De plus, certaines données passées en argument peuvent être modifiées par la fonction appelée.
-

%eax	Nom	Code source	%ebx	%ecx	%edx	%esi	%edi
1	sys_exit	kernel/ exit.c	int				
2	sys_fork	arch/i386/ kernel/ process.c	strucl pt_regs				
3	sys_read	fs/ read_write .c	unsigned int	char *	size_t		
4	sys_write	fs/ read_write .c	unsigned int	char *	size_t		

5	sys_open	fs/open.c	const char *	int	int		
6	sys_close	fs/open.c	unsigned int				
8	sys_create	fs/open.c	const char *	unsigned int			
Figure 3.1: Table de certains appels systèmes Linux							

Dès la fin de la fonction appelée, l'exécution du programme assembleur reprend à l'instruction suivant l'interruption.

Un descriptif plus complet de ces appels système est disponible en annexe E (à la fin de ce document).

4.3 Gestion d'entrée-sortie

- Lecture et écriture
- Ouverture et fermeture d'un fichier

Avant de manipuler des interruptions d'entrée-sortie, il nous faut présenter les notions de *identificateur de fichier* et de *descripteur de fichier*.

- Un identificateur de fichier correspond à une chaîne de caractères ASCII d'au plus 128 octets terminée par un 0. Cette chaîne est composée d'un chemin d'accès et d'un nom de fichier. Ce nom est la seule information non optionnelle. L'identificateur peut être composé de caractères spéciaux (*, etc).
- Un numéro d'accès est associé par le système d'exploitation à chaque fichier. Ce numéro — codé sur 2 octets — permet d'identifier le fichier lors des manipulations. Il est appelé le *descripteur* du flux. De plus, notons que certains périphériques sont gérés comme des fichiers ; ainsi, à l'entrée standard (usuellement le clavier) correspond le numéro d'accès 0 et on a les correspondances :

descripteur	support
0	entrée standard (usuellement le clavier)
1	sortie standard (usuellement l'écran)
2	sortie d'erreur (idem)

4.3.1 Lecture et écriture

Exercice 1
Que fait ce programme?
.data
N :

```
    .byte 3


.bss
BUFF:
        .space 10
.text
.globl _start
_start:

      mov    $3      , %eax
      mov    $0      , %ebx
      mov    $BUFF   , %ecx
      mov    $N      , %edx
      int    $0x80

addl $2, BUFF
mov    %eax    , %edx
      mov    $4      , %eax
      mov    $1      , %ebx
      mov    $BUFF   , %ecx
      int    $0x80

      mov    $1      , %eax
      mov    $0      , %ebx
      int    $0x80
```

Exercice 2 — Entrée et sortie standard.

1. Écrire un programme qui lit au clavier un entier *N* codé sur un caractère (de 0 à 9 donc) et affiche une ligne de *N* caractères '`*`'.
2. Écrire un programme qui lit au clavier un entier *N* codé sur plusieurs caractères (saisie finie par "retour chariot" — code ASCII 13) et affiche une ligne de *N* caractères '`*`'.
3. Écrire un programme qui lit au clavier un entier *N* codé sur plusieurs caractères (saisie finie par "retour chariot") et affiche un carré composé de '`*`' de côté *N*.
4.

4.3.2 Ouverture et fermeture d'un fichier

Exercice 3 — Manipulation de fichiers.

Créer un fichier `foo.essai` et remplissez le avec un identificateur de fichier `bar.essai`. (Pensez à détruire le fichier créé — `rm foo.essai` — si vous voulez recommencer cette opération).
Après avoir créé un fichier contenant un identificateur de fichier, construisez un programme assembleur qui lit ce fichier et place l'identificateur de fichier en mémoire.

En supposant que l'identificateur de fichier précédent n'est pas associé à un fichier existant, construire un programme assembleur qui crée un fichier du même nom et stocke à l'intérieur la chaîne de caractères représentant l'identificateur de fichier ainsi que l'identificateur.

E.1 Entrée – sortie

- sys_close : Fermeture d'un flux
- sys_create : Création d'un fichier
- sys_open : Ouverture d'un flux
- sys_read : Lecture depuis un flux
- sys_write : Écriture dans un flux

- **sys_close : Fermeture d'un flux**
-
- **Description :** Cette fonction ferme le flux de descripteur fd.
- **Entrée(s) :**
-

%eax	6
%ebx	le descripteur du flux.

Sortie(s) :
Prototype C :
```
asmlinkage long sys_close(unsigned int fd)
```

sys_create : Création d'un fichier

Description : Cette fonction crée un fichier de nom filename et retourne le descripteur (un entier) associé.
Entrée(s) :

%eax	8
%ebx	l'adresse de l'identificateur (filename) du fichier
%ecx	le mode (disons $00700 sans plus de détails).

Sortie(s) :

%eax	contient le descripteur du fichier.

Prototype C :

```
asmlinkage long sys_creat(const char __user *filename, int
mode)
```

sys_open : Ouverture d'un flux

Description : Cette fonction ouvre un flux — un fichier — de nom filename et retourne le descripteur (un entier) associé.
Entrée(s) :

%eax	5
%ebx	un pointeur (une adresse) sur la chaîne de caractères représentant le nom du fichier
%ecx	des drapeaux (00 read-only, 01, write only, 10 read-write, 11 special)
%edx	le mode (disons 0 sans plus de détails).

Sortie(s) :

%eax	contient le descripteur du fichier.

Prototype C :

```
asmlinkage long sys_open(const char __user * filename, int
flags, int mode)
```

sys_read : Lecture depuis un flux

Description : Cette fonction récupère %edx octets depuis le flux de numéro %ebx et les place à partir de l'adresse %ecx. Le flux peut être le clavier (descripteur 0) ou un fichier (descripteur retourné par l'appel système sys_open). Cette copie est interrompue dés qu'un retour chariot est rencontré.
Entrée(s) :

%eax	3
%ebx	le descripteur du flux
%ecx	l'adresse de l'espace mémoire où stocker les données lues
%edx	le nombre d'octets à lire.

Sortie(s) :

%eax	contient le nombre d'octets lu.

Prototype C :
```
asmlinkage ssize_t sys_read(unsigned int fd, char __user *
buf, size_t count)
```

sys_write : Écriture dans un flux

Description : Cette fonction écrit %edx octets sur le flux de numéro %ebx en les stockant à partir de l'adresse %ecx. Le flux peut être la sortie standard — généralement l'écran — (descripteur 1) ou un fichier (descripteur retourné par l'appel système sys_open).
Entrée(s) :

%eax	4
%ebx	le descripteur du flux
%ecx	l'adresse de l'espace mémoire contenant les données à écrire
%edx	le nombre d'octets à écrire.

Sortie(s) :

%eax	contient le nombre d'octets écrit.

Prototype C :
```
asmlinkage ssize_t sys_write(unsigned int fd, char __user *
buf, size_t count)
```

sys_exit : Fin de processus

- **Description :** Cette fonction permet de terminer le processus courant en retournant l'entier `error_code` au processus père.
- **Entrée(s) :**

%eax	1
%ebx	un entier error_code

Sortie(s) :

%eax	contient le nombre d'octets lu.

Prototype C : `asmlinkage long sys_exit(int error_code)`