

## TD 3 – Prise en main de l'environnement

### sous LINUX

Vous disposez de 4h de laboratoire pour ce premier TD sous Linux.

But du TD:

Ce TD a comme premier objectif de faciliter le passage de l'environnement de travail MS DOS vers l'environnement de travail Linux.

Le second objectif de ce TD est de poursuivre l'apprentissage des concepts fondamentaux liés aux microprocesseurs tout en utilisant un autre OS<sup>1</sup>. Vous apprendrez à

- écrire des procédures et utiliser le passage de paramètres sur la pile.
- utiliser les appels système du S.E. Linux pour manipuler les fichiers.

La lecture des parties "Interruptions" slide [94 à 98] " et " assembleur sous Linux" des transparents du cours est un pré-requis au présent TD.

### Préalables

#### Environnement

Votre environnement de travail sera en terminal 'texte' sur le serveur *linux1*<sup>2</sup>. Nous reprendrons l'émulation de terminal et le traitement de texte que vous utilisez pour les laboratoires de java : putty et vi.

Vous créerez un répertoire pour les laboratoires assembleur dans votre home directory.

#### Assembleur sous Linux versus assembleur sous Dos

Ci-dessous deux codes assembleur vous sont présentés. Ces deux codes font la même chose (calcul de la surface d'un triangle) mais pour des systèmes d'exploitations différents. Les commentaires associés montrent la différence entre le code écrit pour dos et le source pour Linux. Les différences sont dues au fait que nous n'utilisons ni le même processeur (80386 au lieu de 8086), ni le même système d'exploitation (Linux au lieu de de DOS), ni le même compilateur (nasm au lieu de tasm).

.MODEL SMALL .STACK 100h	global main ; point d'entrée du programme doit être connu par ld.
=====	
.DATA	section .data ; section qui décrit les variables initialisées
=====	
base db 6	base db 6
hauteur dw 7	hauteur dw 7
DEUX equ 2	deux equ 2
=====	
	section .bss ; section qui décrit les variables non initialisées.
=====	
surface dw ?	surface resw 1 ; réserve 1 word pour la variable surface.
=====	
.CODE	section .code ; point d'entrée du programme doit être connu par ld.
=====	
main:	main: ; point d'entrée du programme
mov ax,@data	
mov ds,ax	
mov al,base	mov al,[base] ; [] signifie valeur
mov ah,0	mov ah,0
mul hauteur	mul word [hauteur] ; base*hauteur, word donne la
	; taille de l'opérande.
mov bx,DEUX	mov bx,deux
div bx	div bx
mov surface,ax	mov [surface],ax
mov ebx,0	mov ebx,0 ; argument 0
mov eax,1	mov eax,1 ; appel système N° 1 = exit
int 80h	int 80h ; appel système

Le compilateur choisi sous Linux est **nasm** : le plus proche de **tasm** au point de vue syntaxe que nous ayons trouvé. Le code source pour Linux se trouve dans le répertoire (/home/hal/distri), sous le nom de SurfTri.asm. Sauvez ce source dans votre répertoire sous Linux1.

<sup>1</sup> Il est possible de faire que de l'assembleur sous DOS mais le but et de vous préparer pour le cours de 2ème

<sup>2</sup> Environnement qui vous est bien familier; bonne nouvelle non ?

## Compiler, éditer et exécuter un programme assembleur sous Linux<sup>1</sup>

### Compilation:

```
Compilez votre source : nasm SurfTri.asm -o SurfTri.o -f elf.
```

Le fichier SurfTri.o est au format elf (Exécutable and Linkable Format). C'est le format de fichier binaire standard sous Linux. Un autre format existe pour mémoriser un programme ( format out ) mais il est obsolète.

```
SurfTri.asm-----> Compilation -----> SurfTri.o
```

Notez qu'il est possible de compiler en utilisant gcc, compilateur C et éditeur de liens ( slide 122 du cours)

### Édition de liens:

```
Réalisez l'édition des liens : ld SurfTri.o -o SurfTri -e main
```

Vous devez préciser le point d'entrée de votre programme avec l'option -e. Le main définit la première instruction à exécuter de votre programme.

```
SurfTri.o -----> Edition de liens -----> SurfTri.exe (SurfTri* sous Linux)
```

### Exécution:

```
Exécutez votre programme : ./SurfTri
```

Vous lancez l'exécution en précisant où se trouve votre fichier exécutable. Ici c'est le répertoire courant

## Travailler chez soi

Pour pouvoir travailler chez vous, vous aurez besoins des ressources nécessaires. Consulter le site <http://esi.namok.be>, partie outils Linux, pour préparer votre environnement de travail.

Vous trouverez de l'aide sur les logiciels nasm et ld dans les pages de manuel **man nasm**, via le logiciel info **info nasm** et via le manuel **nasm**, document se trouvant dans la partie ressources du site ([esi.namok.be](http://esi.namok.be))

## Outil de débogage

Sous Linux<sup>1</sup>, vous ne disposerez pas d'un environnement de débogage aussi convivial que celui vu sous Dos<sup>2</sup>. De ce fait, on vous demande d'être très rigoureux dans votre logique. Des erreurs du type « segmentation fault » sont assez fréquentes et difficiles à repérer. En effet les étapes de compilation

<sup>3</sup> Sous Linux il existe **gdb** le débogueur GNU qu'on peut utiliser avec le compilateur gcc (slide 126 du cours).

et d'édition des liens se passent bien mais l'exécution génère une erreur.

Pour faciliter le débogage de votre code, vous trouverez à votre disposition sur le distri (/home/jcj/distri), les fonctions suivantes:

**printw.asm** procédure qui affiche à l'écran un nombre qui se trouve dans un word que vous passez comme argument sur la pile. Ce nombre sera affiché en décimal

**printd.asm** procédure qui affiche à l'écran un nombre qui se trouve dans un double word que vous passez comme argument sur la pile. Ce nombre sera affiché en hexadécimal, en décimal non signé et en décimal signé.

## Appel de procédure

Exemple: utilisation du printw

Pour utiliser efficacement<sup>4</sup> une procédure vous devez effectuer un appel à cette procédure. Dans le cas de la procédure printw :

```
- Ajoutez au début de votre programme SurfTri.asm la directive extern printw afin que le compilateur sache qu'il ne doit pas trouver printw dans votre code.
```

```
- Compilez votre source : nasm SurfTri.asm -o SurfTri.o -f elf.
```

```
- Compilez votre procédure printw : nasm printw.asm -o printw.o -f elf.
```

```
- Réaliser l'édition des liens : ld SurfTri.o printw.o -o SurfTri -e main
```

```
- Exécutez votre programme : ./SurfTri
```

Exercice 1:

- Quelle est la valeur affichée pour la surface du triangle?

- Qu'est ce qu'une procédure ? Et comment ça fonctionne? ( voir slide 96 ..)

Exercice 2:

Écrire le programme parite.asm permettant de vérifier si le contenu d'une variable a ( définie comme un word) est pair ou impair (premier programme du TD2). Votre programme fera appel à la procédure printd pour afficher le résultat sur l'écran.

## Appel système

Le système d'exploitation offre une série de services sous forme de bouts de codes que l'on appelle via une interruption. Ce sont les "appels système". On passe des arguments aux appels système via les registres.

- Les appels système en DOS, ( int 21h par exemple) utilisent **AH** pour préciser l'appel système souhaité. La liste de ces appels système (interruptions) sont indiqués dans helpc

<sup>4</sup> Il est possible d'inclure le code de la procédure dans votre programme ce qu'il la rendra difficilement réutilisable.

- Sous Linux, d'autres appels système existent. Ils sont appelés via int 80h où EAX précise l'appel système souhaité. La liste des appels système et leur numéro se trouvent dans le fichier **/usr/include/asm/unistd.h**.

Les arguments et l'explication de ce qu'ils font se trouvent dans les pages de manuels, chapitre 2. L'explication est donnée en langage C et la liste des arguments suit l'ordre EBX, ECX, EDX, ESI, EDI.

#### Exemple: chmod

Vous avez déjà utilisé la commande chmod pour changer les droits d'accès à un fichier.

Écrivons le code assembleur qui effectue ce changement de droits :

- A l'aide du fichier /usr/include/asm/unistd.h, vous lisez le numéro de l'appel chmod : c'est 15.
- A l'aide de man 2 chmod, vous lisez que les arguments sont, dans l'ordre, le nom du fichier et la nouvelle valeur souhaitée.
- Vous savez que l'interruption correspondant aux appels système sous Linux est 80h.

```
MOV EAX, 15           ; le numéro de l'appel système chmod
MOV EBX, Nom         ; l'adresse de la variable contenant le nom du fichier
MOV ECX, 110110110b ; les nouveaux droits en binaire
INT 80H              ; appel système sous Linux
```

#### Exercice1:

Utilisez le code ci-dessus pour modifier les droits de votre fichier SurfTri.asm

#### Exercice2:

Comment faire pour terminer un processus en Linux ? Vérifiez que vous retrouvez les instructions nécessaires en refaisant le même parcours.

#### Exercice3:

Écrire une procédure permettant d'afficher la valeur des flags. Cette procédure affichera le caractère correspondant au flag si ce dernier est positionné à 1 et le caractère espace ' ' dans le cas contraire. Testez votre procédure.

*Aide* : si seuls les flags c, z, o et p sont à 1 et les autres pas, on aura comme résultat sur le stdout (l'écran)

cz o p

NB: cette procédure sera un outil supplémentaire vous facilitant le débogage de vos codes.

## Manipulation de fichiers

Le système d'exploitation offre une série d'appels système qui permettent au processus de retrouver, de lire ou de modifier les données qui se trouvent sur disque.

- open permet de retrouver un fichier, de l'ouvrir, et de lui associer un descripteur de fichier ouvert contenant la position actuelle dans le fichier (initialement zéro).
- read permet de transférer dans la mémoire, un ensemble de bytes en provenance d'un fichier ouvert, à la position actuelle. La position actuelle est incrémentée du nombre de bytes lus.
- write permet de transférer un ensemble de bytes en provenance de la mémoire vers un fichier ouvert, à la position actuelle. La position actuelle est incrémentée du nombre de bytes écrits.
- close permet de fermer un fichier ouvert.
- lseek permet de modifier la position actuelle d'un fichier.

Valeurs de certaines constantes : (sources : **/usr/include/bits/fcntl.h**)

- O\_RDONLY pour ouvrir un fichier en lecture seulement = 00
- O\_WRONLY pour ouvrir un fichier en écriture seulement = 01
- O\_RDWR pour ouvrir un fichier en lecture et écriture = 02
- O\_CREATE pour ouvrir un fichier en le créant s'il n'existe pas = 0100
- SEEK\_SET, SEEK\_CUR, SEEK\_END (début, position courante, fin du fichier) = 00,01,02.

NB: Les valeurs ci-dessus sont octales

#### Exercice1:

Écrivez le programme créerFichier.asm qui crée le fichier sortie.dat qui contiendra le message << Bienvenue sous Linux1>>

#### Aide:

- Ouvrir le fichier en création, avec ses droits.
- Écrire le texte demandé.
- Fermer le fichier.

#### Exercice2:

Écrivez un programme ajoutInfo.asm qui ajoute au fichier sortie.dat, le message << Ceci est le serveur des lères >>

#### Aide :

- Ouvrir le fichier en écriture.
- Se positionner à la fin du fichier (= déplacement de 0 par rapport à la fin du fichier)
- Écrire le texte demandé.
- Fermer le fichier.

## Manipulation du clavier et de l'écran.

Linux permet d'utiliser les périphériques comme des fichiers. De plus, tout processus ouvre automatiquement 3 fichiers :

- Le descripteur 0, associé normalement au clavier que l'on ne peut que lire.
- Le descripteur 1, associé normalement à l'écran, où l'on ne peut qu'écrire.
- Le descripteur 2, associé normalement à l'écran pour y envoyer des messages d'erreur, où l'on ne peut qu'écrire.

Ces fichiers seront automatiquement fermés à la fin du processus.

#### Exercice1:

Écrivez un programme affichage.asm qui affiche à l'écran <<Hello World>>

#### Exercice2:

Écrivez un programme echo.asm qui affiche à l'écran ce que l'on a frappé au clavier.

NB: La fin d'un fichier est détectée lorsque, lors de l'appel système read, le nombre de caractères reçus est plus petit que le nombre demandés.

Une façon pratique de lire un fichier jusqu'à la fin est d'écrire :

```
Tant que (nb_caractère_lu > 0) Faire
lire le fichier
Fin tant
```

Pour le fichier 'clavier', signaler qu'il n'y a plus rien à lire, se fait par

```
Ctrl-D.
```

#### **la commande cat:**

La commande cat sans argument est une commande qui écrit à l'écran tout ce que l'on tape au clavier.

Exercice3: Écrivez un programme monCat.asm qui effectue la même chose que la commande cat sans argument.

Testez votre programme :

simplement : ./monCat

en redirigeant l'écran, pour créer un fichier : ./monCat > nouveau

en redirigeant le clavier, pour lire un fichier : ./monCat < nouveau

Vous pourriez être étonnés que l'affichage n'est pas synchronisé avec l'appel write. Linux place dans un buffer toutes les écritures afin de les afficher en une seule fois. En général, le saut de ligne provoque le transfert du buffer vers l'écran. Si vous voulez imposer ce transfert, vous pouvez utiliser fflush.

**Exercice d'évaluation** : Écrire un programme qui lit dix nombres (entiers non signés) au clavier et écrit dans le fichier résultat ces mêmes nombres triés au préalable par ordre croissant.

*Aide:* On vous conseille

- de déclarer dans votre section .bss un tableau de 10 éléments
- de faire une boucle qui lit les nombres et les places dans le tableau
- de trier le tableau avec un algorithme de tri simple
- d'écrire dans le fichier résultat, les nombres séparés par des espaces ou par des sauts de lignes.