

## 1 But du td04

Premier contact avec l'environnement de développement de travail en terminal 'texte' sur le serveur linux1. Nous reprendrons l'émulation de terminal et le traitement de texte que vous utilisez pour les labos de java : putty et vi. Nous écrirons des procédures et utiliserons le passage de paramètres sur la pile. Nous utiliserons quelques appels système du S.E. Linux pour manipuler les fichiers.

## 2 Compiler et exécuter un programme assembleur sous Linux1

A l'aide de vi, visualiser la source assembleur donnée dans mon distri (/home/jcj/distri) sous le nom de td04a.asm. Elle reprend le calcul demandé dans td01 pour fournir le volume d'un cône. Les commentaires associés montrent la différence entre la source écrite sous dos et la 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). Le compilateur choisi sous linux est nasm : le plus proche de tasm au point de vue syntaxe que nous avons trouvé. Sauvez cette source dans votre répertoire.

Compilez votre source : `nasm td04a.asm -o td04a.o -f elf.`

Réalisez l'édition des liens : `ld td04a.o -o td04a -e main`

Exécutez votre programme : `./td04a`

1. Le compilateur transforme votre source td04a.asm en objet td04a.o au format elf. Linux utilise deux formats pour mémoriser un programme : out et elf. Le format out est obsolète.
2. L'éditeur de liens transforme votre objet td04a.o en exécutable td04a. Vous devez préciser le point d'entrée de votre programme avec l'option -e. C'est l'adresse de la première instruction à exécuter de votre programme.
3. Vous lancez l'exécution en précisant où se trouve votre fichier exécutable.

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 internet `google/linux, lea, ...`.

Vous trouverez dans mon distri, réécrits pour Linux, certains des exercices demandés lors des td02 et 03 sous les noms td02 ?.asm et td03 ?.asm.

### 3 Appels de procédures

Rappels de ce qui a été vu au cours MIC et système d'exploitation :

1. CALL "adresse" est une instruction qui mémorise IP sur la pile (IP pointe à ce moment vers l'instruction qui suit le CALL) et qui remplace IP par "adresse". La prochaine instruction exécutée est donc celle qui se trouve à "adresse".
2. RET est une instruction qui remplace IP par l'information qui se trouve au sommet de la pile. La prochaine instruction exécutée est donc celle qui se trouve à "adresse" qui se trouvait au sommet de la pile.

L'ensemble CALL "adresse" et RET permet d'exécuter un code qui se trouve entre "adresse" et RET que l'on appelle procédure, le nom donné à cette procédure est souvent celui donné à "adresse".

Lorsqu'on souhaite transmettre des arguments à une procédure, on utilise les registres ou la pile. Avant d'appeler la procédure, on place les arguments sur la pile à l'aide de l'instruction PUSH. La procédure ira rechercher ces arguments sur la pile.

Exercice 1 :

Reprenez le code printw qui se trouve dans mon distri (printw est une procédure qui affiche à l'écran le chiffre qui se trouve dans le word que vous passez comme argument sur la pile). Ajoutez ce code au programme td04a.asm pour en faire td04b.asm de telle façon que le volume du cône soit affiché à l'écran.

Vous pourriez utiliser printw dans plusieurs de vos programmes. Pourquoi multiplier ce code en autant de programme ? Si vous deviez modifier le code printw, vous devriez le faire dans chaque programme ! Il est beaucoup plus simple de réutiliser une version compilée de printw :

Modifiez td04a.asm qui effectue l'appel à la procédure que vous n'incluez pas pour en faire td04c.asm. Ajoutez au début «extern printw» afin que le compilateur sache qu'il ne doit pas trouver printw dans votre code.

Compilez votre source : `nasm td04c.asm -o td04c.o -f elf.`

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

Réalisez l'édition des liens : `ld td04c.o printw.o -o td04c -e main`

Exécutez votre programme : `./td04c`

## 4 Appels système

Rappels de ce qui a été vu au cours de système d'exploitation : Le système d'exploitation offre une série de services sous forme de bout de code que l'on appelle via une interruption. Ce sont les "appels système". On passe des arguments aux appels système via les registres. Ceux-ci renvoient des informations via le registre EAX. Si plus d'une information doit être renvoyée, quelques arguments fournissent l'adresse de mémoires où l'appel système peut enregistrer ces renseignements complémentaires.

Vous avez déjà utilisé les appels système en DOS, essentiellement int 21h où AH précise l'appel système souhaité.

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 1 : Vous avez déjà utilisé chmod pour changer les droits d'accès à un fichier. Ecrivons le code assembleur qui effectue ce changement de droits :

1. A l'aide du fichier /usr/include/asm/unistd.h, vous lisez le numéro de l'appel chmod : c'est 15.
2. 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.
3. Vous savez que l'interruption correspondant aux appels système 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
```

Exemple 2 : Pour terminer un process en DOS vous utilisez l'appel système AH=4C et int 21H. C'est l'exit d'un process. Comment faire en Linux ?

1. A l'aide du fichier /usr/include/asm/unistd.h, vous lisez le numéro de l'appel exit : c'est 1.
2. A l'aide de man 2 exit, vous lisez que le seul argument est le numéro de l'exit que vous pourrez lire après la fin du programme avec echo \$?.
3. Vous savez que l'interruption correspondant aux appels système est 80h.

```
MOV EAX, 1      ; le numéro de l'appel système exit
MOV EBX, 12     ; le code de sortie du process
INT 80H        ; appel système
```

Exercice : Ecrivez un programme td04g.asm qui modifie les droits d'accès d'un de vos fichier.

## 5 Manipulation de fichiers

Rappels de ce qui a été vu au cours de système d'exploitation : Le système d'exploitation utilise une stratégie pour enregistrer des données sur un disque ou tout autre support. Nous avons étudié le système FAT et le système NTFS qu'utilise les systèmes de Microsoft. Linux utilise d'autres stratégies et celle utilisée sur Linux 1 est le système ext2. Ces 3 systèmes offrent une vue hiérarchisée d'un système de fichiers.

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

1. 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. Ici, zéro.
2. 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.
3. 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.
4. close permet de fermer un fichier ouvert.
5. lseek permet de modifier la position actuelle d'un fichier.

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

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

Exercice : Ecrivez un programme td04k.asm qui crée un fichier td04k.dat qui contiendra «Hello World»

Petite aide :

1. Ouvrir le fichier en création, avec ses droits.
2. Ecrire le texte demandé.
3. Fermer le fichier.

Exercice : Ecrivez un programme td04l.asm qui ajoute au fichier td04k.dat, «En français, Bonjour le Monde»

Petite aide :

1. Ouvrir le fichier en écriture.
2. Se positionner à la fin du fichier (= déplacement de 0 par rapport à la fin du fichier)
3. Ecrire le texte demandé.
4. Fermer le fichier.

## 6 Manipulation du clavier et de l'écran.

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

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

Ces fichiers seront automatiquement fermés à la fin du process.

Exercice : Ecrivez un programme td04p.asm qui affiche à l'écran «Hello World»

Exercice : Ecrivez un programme td04q.asm qui affiche à l'écran ce que l'on a frappé au clavier.

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 le nombre de caractères lus est positif, lire le fichier.

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

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

Exercice : Ecrivez un programme td04r.asm qui effectue la même chose que la commande cat sans argument.

Testez votre programme :

1. simplement : ./td04r
2. en redirigeant l'écran, pour créer un fichier : ./td04r > nouveau
3. en redirigeant le clavier, pour lire un fichier : ./td04r < 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 : Utilisez les pages de manuel pour savoir ce que fait la commande head. Réécrivez la commande head sans argument, en assembleur : td04s.asm.