

## 1 But du td05

Comprendre le passage de paramètres sur la pile pour un exécutable. Comprendre le passage de paramètres sur la pile pour une procédure. Comprendre les variables locales.

## 2 Utilitaires Linux

Lors du td04, nous avons étudié le fonctionnement des utilitaires tels que `cat` ou `head`, mais sans argument. Si on ne donne aucun argument, la plupart des utilitaires utilisent le clavier (input standard) en lecture et l'écran (output standard) en écriture.

Si on donne des arguments, la plupart des utilitaires considèrent que ces arguments sont des noms de fichiers et que ceux-ci doivent être, chacun à leur tour, considérés comme l'input standard.

A titre d'exemple, vérifiez, en ligne de commande, le comportement de l'utilitaire `cat`, sans, avec un argument, avec 3 arguments. (Si vous avez oublié ce que fait `cat` : `man cat`).

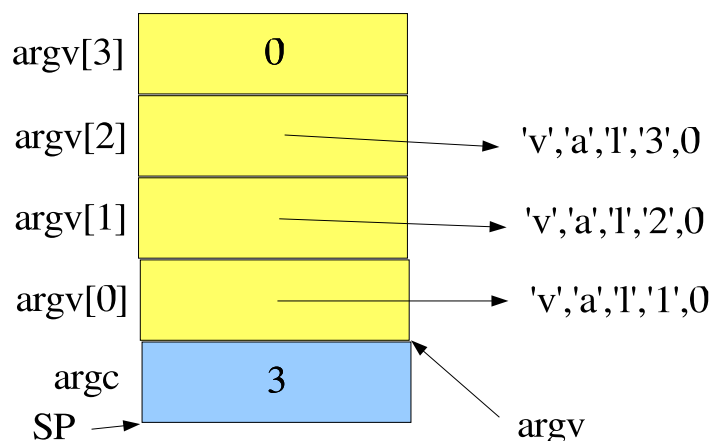
La plupart des utilitaires de Linux sont écrits suivant la logique : (exemple du `cat`)

```
traitement(h)
{ Lire un caractère sur la handle h
  Tant que le nombre de caractères lus = 1
    Ecrire un caractère sur le handle 1 (écran).
    Lire un caractère sur la handle h
}

main ( argc, argv )
{ Si argc=1 Alors traitement(0);
  Sinon Pour i allant de 1 à argc-1
    ouvrir le fichier argv[i] dans le handle h
    traitement(h)
    Fermer(h);
  Terminer le programme
}
```

Les arguments du main, argc et argv sont passés sur la pile. Le dessin suivant explique comment. Pourquoi ne trouve t'on pas d'adresse de retour (IP) sur la pile ?

Etat de la pile au debut de main, si on a appelé le programme par ./programme val1, val2 val3  
Toutes ces valeurs dans la piles sont des 32 bits ou 4 bytes.



Exercice : Ecrivez un programme td05a.asm qui effectue la même chose que la commande echo.

Vous testerez votre programme avec les commandes suivantes :

```
./td05a Hello World
./td05a *.asm
./td05a Hello World > Nouveau
```

Exercice : Ecrivez un programme td05b.asm qui effectue la même chose que la commande head.

Vous testerez votre programme avec les commandes suivantes :

```
./td05b
./td05b td05b.asm
./td05b *.asm
./td05b > Nouveau
```

### 3 Procédures, passage d'arguments et valeur de retour.

On peut passer des arguments à une procédure en plaçant ces arguments dans des registres. C'est la technique utilisées pour les appels système. Mais le nombre et la taille des registres sont limités.

On peut passer des arguments à une procédure en plaçant la valeur de ces arguments dans la pile. Ces arguments sont lus par la procédure mais ne peuvent être modifiés car la procédure ne connaît pas l'adresse de ces arguments. (on parle de passage d'arguments par valeur)

On peut passer des arguments à une procédure en plaçant l'adresse de ces arguments dans la pile. Ces arguments peuvent être lus par la procédure mais peuvent aussi être modifiés car la procédure connaît l'adresse de ces arguments. (on parle de passage d'arguments par adresse)

La procédure peut renvoyer une information au programme qui l'appelle. En général, cette information se trouve dans le registre EAX. C'est la valeur de retour. Si la procédure doit renvoyer plus d'une information, on lui passe, en argument, l'adresse où elle peut fournir ces informations.

Exercice : Ecrivez une procédure affiche qui affiche à l'écran une chaîne de maximum 80 caractères terminée par un zéro binaire. Elle termine son affichage par un saut de ligne. Testez votre procédure avec un programme td05c.asm qui appelle affiche comme procédure externe. La fonction affiche utilise comme argument l'adresse de la chaîne à afficher et renvoie (EAX)

- 0 : si pas d'erreur.
- 1 : si plus de 80 caractères à afficher.

Exercice : Ecrivez une procédure itoa.asm qui transforme un nombre binaire contenu dans un double word en une chaîne de caractère terminée par un zéro binaire. Testez votre procédure avec un programme td05d.asm qui appelle itoa et affiche comme procédures externes. La fonction atoi utilise comme arguments

- la valeur du nombre à transformer
  - le type du nombre à transformer (0 = nombre non signé, 1 = nombre signé)
  - la base dans laquelle le nombre doit être affiché ( 2=binaire, 8=octal, 10=décimal, 16=hexadécimal)
  - la taille du résultat ( 0=longueur du chiffre, n=longueur demandée)
  - l'adresse de la chaîne de caractère où doit être mémorisé le résultat
- et renvoie (EAX)
- 0 : pas d'erreur
  - 1 : la base n'est pas correcte : chaîne = 'xxx',0
  - 2 : le type n'est pas correct : le nombre est considéré non signé

- 3 : le nombre ne peut être transformé en n caractères (n est trop petit) :  
taille=0

Exercice : Ecrivez une procédure lit qui lit au clavier une ligne et qui la transforme en une chaîne de maximum 80 caractères terminée par un zéro binaire. Testez votre procédure avec un programme td05e.asm qui appelle lit et affiche comme procédure externe. La fonction lit utilise comme argument l'adresse de la chaîne résultat de la lecture et renvoie (EAX)

- 0 : si pas d'erreur.
- 1 : si plus de 80 caractères ont été lus : tronquer à 80 caractères.

Exercice : Ecrivez une procédure atoi.asm qui transforme une chaîne de caractère terminée par un zéro binaire en un nombre binaire contenu dans un double word. La chaîne est au format [+/-][digit] Testez votre procédure avec un programme td05f.asm qui appelle lit, atoi, itoa et affiche comme procédures externes. La fonction itoa utilise comme arguments

- l'adresse de la chaîne de caractère où doit être mémorisé le résultat
- le type du nombre à transformer (0 = nombre non signé, 1 = nombre signé)
- la base dans laquelle le nombre doit être transformé ( 2=binaire, 8=octal, 10=décimal, 16=hexadécimal)
- l'adresse du nombre transformé

et renvoie (EAX)

- 0 : pas d'erreur
- 1 : la base n'est pas correcte (nombre=0)
- 2 : le type n'est pas correct (le nombre est considéré non signé)
- 3 : le nombre est trop grand. (nombre=0)
- 4 : la chaîne contient des caractères invalides. (nombre=0)

Exercice d'évaluation : Ecrivez un programme td05g.asm qui lit deux nombres hexadécimaux au clavier, en calcule la somme, et affiche le résultat à l'écran en binaire, en octal, en décimal et en hexadécimal. Toutes les erreurs éventuelles seront affichées sur stderr. Ce programme utilisera comme procédures externes : lit, affiche, atoi et itoa.