TD 4 – Langage d'assemblage sous Linux avec nasm (seconde partie)[1]

Manipulation de fichiers, récupération d'arguments par le biais de la ligne de commande, transmission d'arguments à une procédure par la pile, valeur en retour d'une procédure, variables locales.

Date de remise de l'énoncé : semaine du 7 avril 2008 (30).

Date de l'évaluation : semaine du 5 mai 2008 (34).

Utilitaires Linux

Dans le TD03, le fonctionnement de la commande `cat` *sans* argument a été étudié. Lorsqu'on ne fournit pas d'argument, la plupart des utilitaires utilisent les entrée et sortie standard (la console, donc).

Si on donne des arguments, la plupart des utilitaires considèrent qu'il s'agit de noms de fichiers et que ces derniers doivent être traités séquentiellement et comme l'entrée standard.

À titre d'exemple, vérifiez, en ligne de commande, le comportement de `cat` sans, avec un puis avec trois arguments.

La plupart des utilitaires Linux sont écrits selon la logique suivante :

```
main(argc, argv)
{
    si (argc == 1)
        traiter(0)          /* clavier */
    sinon
        pour i de 1 à argc – 1
            ouvrir le fichier argv[i] dans le handle h
            traiter(h)
            fermer(h)
        fpour
    fsi
    fin du programme
}

traiter(i)
{
    /* ici traitement propre à la commande */
}
```

1 Ce TD s'inspire largement du td05 rédigé par JCJ en 2006.

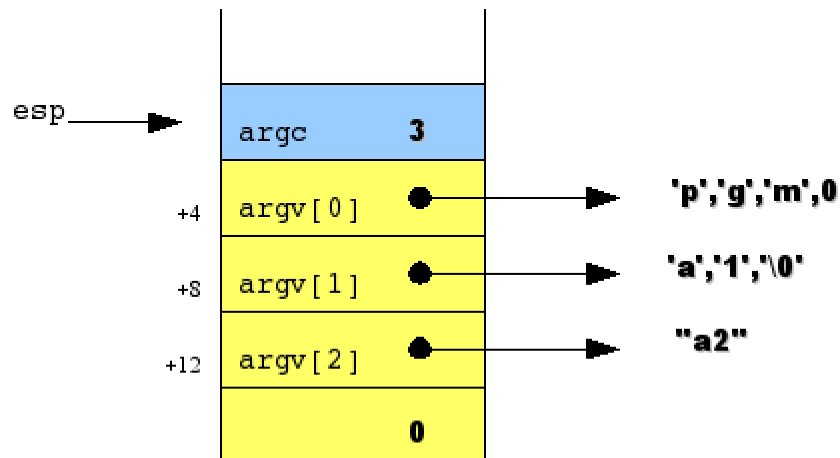

Fig. 1: État de la pile après invocation de `pgm` *avec les arguments* `a1` *et* `a2` *:* `./pgm a1 a2` ↵
`pgm` *a été produit avec* `ld` *avec l'option* `-e` *(le nom de la fonction principale est quelconque).*

Pour la commande `cat`, la fonction de traitement est :

```
traiter(i)
{
    lire un caractère sur le handle i
    tant que (nombre de caractères lus == 1)
        écrire un caractère sur le handle 1        /* écran */
        lire un caractère sur le handle i
    ftq
}
```

Les arguments de la fonction principale du programme sont transmis par la pile. Les figures 1 et 2 expliquent comment. La situation décrite dans la Fig. 1 apparaît lorsque l'éditeur de liens utilisé est `ld`. Celle de la Fig. 2 résulte de l'utilisation de `gcc`, avec un point d'entrée nommé `main`.

Notez que, sur ces figures, la pile se remplit *vers le haut*. Le registre `esp` pointe, comme toujours, sur (le premier octet de) la dernière valeur empilée. Toutes les valeurs dans la pile et tous les pointeurs ont comme taille 32 bits (ou 4 octets).

Mise en pratique

1. Écrivez un programme, `nEcho`, qui effectue la même chose que la commande `echo` sans option. Remarquez que cette commande ne se comporte *pas* comme l'algorithme donné ci-dessus. Testez votre programme avec les commandes suivantes :

   a) `./nEcho Hello World;`
   b) `./nEcho Hello World > fichierSortie`.
   c) `./nEcho *.asm`.

2. Écrivez un programme `nRev` qui effectue le même travail que la commande `rev`.

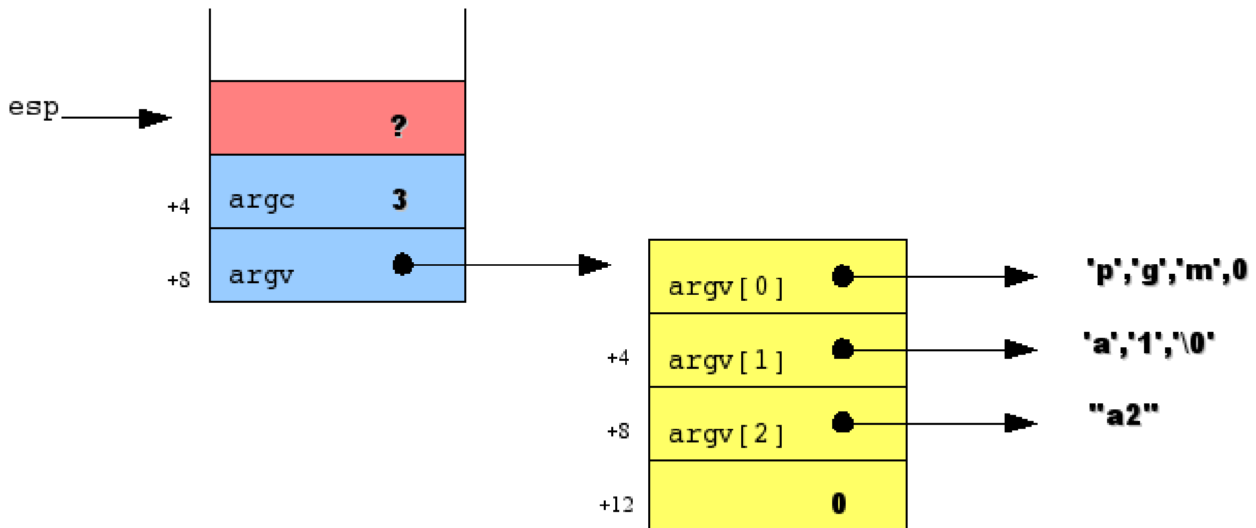

Fig. 2: État de la pile après invocation de `pgm` *avec les arguments* `a1` *et* `a2` *:* `./pgm a1 a2 ↵`
`pgm` *a été produit avec* `gcc` *sans l'option* `-e` *(le nom de la fonction principale est* `main`*).*

3. Écrivez un programme, `nTail`, qui effectue la même chose que la commande `tail` sans option. Testez votre programme avec les commandes suivantes :

   a) `./nTail;`
   b) `./nTail nTail.asm;`
   c) `./nTail *.asm;`
   d) `./nTail > fichierSortie.`

4. Écrivez le programme `nPaste` identique à la commande `paste` sans option.

Procédure, passage d'argument, valeur de retour et variable locale

On peut passer les arguments d'une procédure en les plaçant dans des registres. C'est la technique utilisée pour les appels système. Cependant, le nombre et la taille des registres sont limités.

On peut passer des arguments à une procédure en plaçant la *valeur* de ses arguments sur la pile. Elles sont lues par la procédure, mais les éventuelles *variables* dont elles sont issues ne peuvent être modifiées car la procédure ne connaît pas l'*adresse* de ces variables. On parle dans ce cas d'une *transmission d'argument par valeur*.

On peut passer des arguments à une procédure en plaçant l'*adresse* de variables sur la pile. Les valeurs de ces variables peuvent être lues par la procédure. Elles peuvent aussi être modifiées au sein de la procédure car cette dernière connaît l'adresse de ses arguments. On parle de *passage d'arguments par adresse*. Remarquez que l'adresse d'une variable peut également être transmise par registre, et qu'il s'agit d'une valeur[1].

La procédure peut renvoyer une information au programme qui l'appelle. En général[2], cette

1 Stockable dans un pointeur.
2 Si c'est un entier ou une adresse.

information est transmise via le registre `eax`. C'est la valeur de retour. Si la procédure doit renvoyer plus d'une information, on lui passe, en argument, la ou les adresses où elle peut fournir ces informations.

La pile sert également, au sein d'une procédure, à y créer des variables locales. Leur création, sous la forme de *trou* dans la pile, et leur destruction en fin de procédure n'est *pas* automatique en langage d'assemblage.

Dans *tous* les exercices qui suivent, veillez à :

- transmettre tous les arguments par la *pile* ;
- n'utiliser *aucune* variable globale, à l'exception des chaînes de caractères à afficher.

Mise en pratique

5. Écrivez une procédure `afficher` qui affiche à l'écran une chaîne de maximum 80 caractères terminée par un zéro binaire. L'affichage de la chaîne peut être suivi par un saut de ligne, selon les désirs de l'utilisateur. Testez votre procédure avec un programme `testAfficher.asm` qui appelle `afficher` comme procédure externe. La fonction `afficher` utilise comme arguments :
   a) l'adresse de la chaîne à afficher ;
   b) une valeur numérique stockée sur un double mot : si cette valeur est non nulle, l'affichage de la chaîne est suivi par un passage à la ligne.

   Elle renvoie dans `eax` :
   a) 0 : si aucune erreur n'est rencontrée ;
   b) 1 : si on tente d'afficher une chaîne de plus de 80 caractères.

6. Écrivez une procédure `entierBinToStr` qui transforme un entier stocké dans un double mot en une chaîne de caractères terminée par un zéro binaire. Testez votre procédure avec un programme `testEBTS` qui appelle `entierBinToStr` et `afficher` comme procédures externes. La fonction `entierBinToStr` a comme arguments :
   a) la valeur du nombre à transformer ;
   b) la manière d'interpréter le motif binaire du nombre à transformer : 0 pour un nombre non signé, 1 pour un nombre signé ;
   c) l'adresse de la chaîne de caractères où stocker le résultat de la conversion.

   Elle représente toujours l'entier en base 10. Aucune validation relative à la longueur de la chaîne fournie ne peut être réalisée par `entierBinToStr`. Veillez donc à ce que cette chaîne soit suffisamment grande lors de son test (et de ses utilisations ultérieures).

7. Écrivez le programme `nWc` qui réalise le même traitement que la commande `wc` sans option.

Exercice d'évaluation

Écrivez un programme, `palindrome`, qui, pour chaque fichier qu'on lui fournit en entrée (ou l'entrée standard par défaut), compte et affiche, à la suite de son nom (`stdin` pour l'entrée standard), le nombre de lignes palindromes qu'il contient. Après l'affichage relatif à chaque fichier, le nombre total de lignes palindromes est affiché. En outre, si l'option '`-sp`' est activée, les lignes palindromes de chaque fichier sont affichées entre le nom du fichier et leur nombre. Pour vous simplifier la vie, considérez que l'option doit obligatoirement suivre directement le nom du programme (et donc précéder tout autre argument). N'oubliez pas d'afficher un message d'erreur (seule variable globale tolérée) en cas de problème (fichier inexistant, par exemple).