Cours 6
Les informations passées par le S.E. au processus

1 Les arguments de la ligne de commande

Quand l'utilisateur tape une commande dans un interpréteur de commande (ou quand l'interface graphique la génère automatiquement), celle-ci comporte souvent plusieurs mots après le nom du programme (des **arguments**). Ex :
> cc –o mymain mymain.c util.c
Le premier mot est destiné au S.E. qui va exécuter un programme de nom cc. Le reste n'a pas d'intérêt pour le S.E., mais seulement pour le programme. Le passage des arguments de la ligne de commande dépend à la fois du S.E. et du langage :

- le S.E. (Unix ou Windows) découpe la ligne de commande en mots et copie ces mots dans une zone mémoire accessible au processus.
- Le code de démarrage du processus rend ces données accessibles selon les règles du langage employé. En C, œ code calcule le nombre d'arguments, crée un tableau avec les adresses des arguments et passe ces deux données sur la pile.

Pour lire la ligne de commande dans un programme C, on utilise main() sous la forme :
`int main(int, char*[]) ; /*` ou `void main(int, char*[]) ; */`
Si on appelle
`main(int argc, char *argv[]) {……..}`
argc contiendra le nombre d'arguments, y compris le nom du programme. argv[0] est l'adresse d'une chaîne de caractères contenant le nom du programme, argv[1] l'adresse du mot suivant, …, argv[argc –1] l'adresse du dernier argument.

Ce dispositif est très utile pour fournir au programme des chaînes de caractères (par exemple des noms de fichier) qui ne sont pas notés "en dur" dans le code, donc qui peuvent changer d'une exécution à l'autre, sans avoir besoin de les demander interactivement à l'utilisateur. Par exemple, si l'on code :

```
/* Fichier imprime.c */
    main() {
            char nomFic[128] ;
            printf("quel fichier voulez-vous imprimer ?") ;
            scanf "%s",&nomFic) ;
            envoieImprimante(nomFic);
    }
```

on ne peut exécuter imprime quand l'utilisateur n'est pas là. Si l'on code à la place :

```
    int main(int argc, char *argv[])  {
            if (argc = 1) return(1) ; /* il manque le nom du fichier à imprimer */
            envoieImprimante(argv[1]) ;
    }
```

on peut faire exécuter imprime en tâche différée, par ex. la nuit quand tout le monde est parti.
Cette technique a de très nombreuses applications, et on verra qu'elle est utilisée systématiquement dans les scripts. On verra aussi que, quand main() renvoie un int, cette valeur peut être utilisée dans le script pour vérifier si le programme s'est terminé correctement ou en erreur.

2 Les variables d'environnement.

Sur n'importe quel système, un grand nombre d'informations indispensables à l'exécution des processus varient selon la machine et l'utilisateur. Par exemple les exécutables ne sont pas toujours au même endroit (cela dépend entre autres de la place disque), chaque utilisateur a un catalogue de login différent, l'imprimante la plus proche dépend de la salle où l'on est, etc. L'ensemble de ces informations constitue l'environnement du processus.
La plupart des systèmes utilisent pour transmettre ces informations aux processus des variables d'environnement. La technique est la suivante (avec les commandes Unix) :

- l'interpréteur de commandes peut créer et positionner des variables d'environnement interactivement grâce à la commande setenv. Le nom de la variable est une chaîne de caractères, sa valeur une deuxième chaîne. Ex :
  > setenv PRINTER R202lj
  crée ou modifie la variable PRINTER. Sa valeur est R202lj.

- On peut utiliser une variable d'environnement dans une commande en ajoutant un $ devant son nom. Ex. :
  > setenv SOURCE mymain.c util.c
  > emacs $SOURCE #ou aussi emacs ${SOURCE}

- On peut accéder à l'environnement dans un programme C. Lors du lancement du processus, le système transmet sur la pile l'adresse de l'environnement actuel, sous la forme d'un argument de la procédure main(). On utilise pour le lire l'appel :
  main(int argc, char* argv[], char* arge[]),
  où arge est un tableau de chaînes de caractères contenant les variables d'environnement, et terminé par le pointeur NULL. Le processus peut alors récupérer les informations qui l'intéressent. On affiche par exemple tout l'environnement par :

```
main(int argc, char* argv[], char* arge[])  {
        int i=0 ;
        while (arge[i] != NULL) printf("%d\n", arge[i++] ) ;
}
```

  La fonction de bibliothèque getenv() permet de récupérer directement la valeur d'une variable d'environnement. On peut améliorer le code de imprime.c en écrivant :

```
int main(int argc, char *argv[], char *arge[])  {
        char defaut[]="R100lj" ; char  * printer ;
        printer = getenv("PRINTER");   /* lit la valeur de PRINTER dans l'environnement */
        if (printer ==NULL)  printer =defaut ;   /* si cette valeur n'est pas définie, utilise R100lj */
        if (argc = 1) return(1) ;     /* il manque le nom du fichier à imprimer */
        envoieImpimante(argv[1], printer ) ;  /* c'est une nouvelle version de envoieImprimante */
}
```

3 L'entrée et la sortie standard, la sortie erreur

La notion d'entrée standard (stdin), sortie standard (stdout) et sortie erreur (stderr) est une invention d'Unix qui est pour beaucoup dans son succès. Quand un processus est lancé, le S.E. ouvre 3 fichiers avant de lancer la fonction main(). Ces trois fichiers sont stdin, stdout, stderr. Le programmeur n'a donc pas besoin de les ouvrir pour les utiliser. Par contre, il ne sait pas au moment où il écrit son programme à quoi correspondent réellement ces fichiers: on peut changer leur affectation dans la ligne de commande, sans toucher au code déjà compilé. Si la ligne de commande ne comporte pas de redirection, stdin, stdout et stderr sont positionnés sur le clavier et l'écran du terminal d'où le processus est lancé. un signe > après la commande redirige stdout et stderr, < après la commande redirige stdin. Ex.:

```
/* Fichier indente.c */
main() {
char c ;
while((c=getchar()) != EOF) {        /* lit un caractère sur l'entrée standard */
        putchar(c) ;                 /* le recopie sur la sortie standard */
        if (c==EOL) putchar('\t');   /* ajoute une tabulation en tête de ligne (EOL = End Of Line) */
}
```

utilisation (le # est le signe de remarque de l'interpréteur de commande)

```
>indente                    # lit le clavier, affiche à l'écran avec indentation
>indente < mymain.c         # lit mymain.c et l'affiche à l'écran avec indentation ; le clavier n'est pas lu
> indente > comment.txt     # lit le clavier, indente et écrit le résultat dans comment.txt ; rien à l'écran
> indente < brut.txt > propre.txt # écrit dans propre.txt une copie indentée de brut.txt
```