M3101 – Principes des systèmes d'exploitation Semestre 3 DUT IUT de Blagnac	Iulian Ober Patricia Stolf

TP4 (3h) : Accès aux fichiers

Objectif. L'objectif de ce TP est de vous familiariser avec quelques fonctionnalités de base de l'API système POSIX pour la lecture/écriture des fichiers

Rappel sur la documentation. Les fonctions du module **os** en Python sont des copies conformes des fonctions système en langage C, qui disposent de pages manuel UNIX (**man**). La documentation du module Python **os** étant souvent assez succincte, il est utile de consulter également les pages **man** des fonctions C utilisées.

Ex. 1. Lecture/écriture d'un fichier avec *open/read/write/close*.

A l'aide des fonctions **open, read, write** et **close,** écrire un programme Python qui reproduit le fonctionnement de la commande

cat fichier

(i.e., il lit le contenu du fichier et l'affiche à la sortie standard).

Le programme récupère le nom du fichier passé par la ligne de commande via **sys.argv**. L'affichage à la sortie standard doit se faire avec la fonction **write** (et non avec **print**).

A noter :

1. *Les fonctions* **open/read/write/close** *du module* **os** *utilisent les descripteurs de fichiers numériques (entiers) de la table de fichiers du processus. Ainsi, par exemple, le descripteur de la sortie standard est le n° 1 ; il peut être obtenu également via l'appel* **sys.stdout.fileno()** *(qui renvoie donc 1).*
2. *Il existe aussi une fonction built-in (disponible par défaut) appelée* ***open.*** *Vous* ***ne devez pas l'utiliser****, mais utiliser en revanche la fonction* ***open*** *du module* ***os !***
3. *Il convient de traiter les erreurs potentielles à chaque appel système (fichier inexistant, etc.) et afficher une description de la cause de l'erreur.*
   Si vous souhaitez traiter plusieurs appels système avec un seul block **except***, utilisez la fonction* **traceback.print_exc()** *pour afficher la ligne de l'appel ayant causé l'erreur.*

Ex. 2. Variation sur le même thème

Modifier le programme précédent pour que maintenant il reproduise le fonctionnement de la commande

cat > fichier

i.e., il lit des caractères au clavier et les écrit dans un fichier dont le nom est passé en argument.

A noter :

- ***a.*** *Le programme doit être lancé sans redirection (>) et il récupère le nom du fichier passé par la ligne de commande toujours au moyen* ***sys.argv****).*
- ***b.*** *Vous devez utiliser* ***os.read*** *pour lire l'entrée standard (et non pas* ***input****).*

Ex. 3. Sauvegarde et modification d'un fichier

Ecrire un programme qui prend en argument le nom d'un fichier texte **f** et :

- Crée une copie de sauvegarde de **f**, intitulée **f~** (écrase le fichier **f~** si celui-ci existe)
- Supprime les espaces se trouvant *au début* de chaque ligne du fichier **f**

Indication : *la modification du fichier* ***f*** *étant difficile dans ces conditions, un moyen plus simple d'arriver au résultat est de changer le nom de* ***f*** *en* ***f~*** *(en utilisant les fonctions* ***link*** *et* ***unlink****) et de recréer* ***f*** *à partir de* ***f~*** *(en utilisant* ***read/write****) et en supprimant les espaces au passage.*

A noter : *lors de la recréation de* ***f****, il faut veiller à garder les mêmes droits d'accès que sur l'ancien fichier. Utiliser la fonction* ***stat*** *pour les obtenir avant de l'effacer.*

Ex. 4. Un utilitaire pour suivre les appels système

Lancer le programme de l'exercice précédent à l'aide de la commande strace. Donner la liste des différentes fonctions système utilisées par votre programme. (La curiosité devrait vous pousser à consulter les pages de manuel au moins pour certaines d'entre elles !)