

Travaux Dirigés Programmation Système: Feuille 3

Informatique 2ème année. ENSEIRB 2014/2015

—Mathieu Faverge - mfaverge@enseirb.fr —

Gestion de fichiers avec les API POSIX et C standard et création de processus

►Exercice 1. Appels système et libc :

Les fonctions suivantes sont-elles des appels système ou des fonctions de la bibliothèque C standard ?

`printf()`, `fopen()`, `fclose()`, `fread()`, `open()`, `close()`, `lseek()`, `rewind()`, `write()`, `exit()`, `_exit()`

Aidez-vous des prototypes de ces fonctions pour répondre à cette question.

►Exercice 2. libc et buffers :

Le but de cet exercice est de mettre en évidence l'effet des buffers (tampons) sur les entrées-sorties.

Le schéma du programme que vous allez écrire est le suivant :

1. afficher une chaîne de caractères de début
2. mettre le processus en attente une seconde avec `sleep(3)`
3. afficher une chaîne de caractères de fin
4. quitter

Afin de n'avoir qu'un seul programme source, vous allez utiliser le préprocesseur C pour conditionner le fonctionnement de votre programme.

- la chaîne de caractères de début sera contenue dans la variable du préprocesseur (macro) `CHaine1`
- la chaîne de caractères de fin sera contenue dans la variable du préprocesseur `CHaine2`.
- la première écriture se fera avec `fprintf(3)` si `UTILISER_FPRINTF1` est positionnée, et avec `write(2)` sinon
- la deuxième écriture sera conditionnée de la même manière par `UTILISER_FPRINTF2`
- la première écriture se fera sur la sortie d'erreur standard si `UTILISER_SORTIE_ERREUR1` est positionnée, sur la sortie standard autrement
- la deuxième écriture sera conditionnée de la même manière par `UTILISER_SORTIE_ERREUR2`
- le programme quittera par `_exit(2)` si `UTILISER_EXIT` est positionnée, et par `exit(3)` sinon.

Afin de voir les différents modes de fonctionnement des tampons, le programme est exécuté de deux manières différentes :

- `./programme` afin que les sorties soient sur le terminal
- `./programme 2>&1 | cat -u` afin que les sorties du programme passent par un tube avant d'être affichées

Mettez en évidence les trois modes de fonctionnement des buffers de la bibliothèque C standard, ainsi que le fait que les appels système n'utilisent pas de buffers visibles au niveau utilisateur.

Testez notamment les modes de compilations suivants :

- `-DUTILISER_FPRINTF1`
- `-DUTILISER_FPRINTF2 -DUTILISER_SORTIE_ERREUR2`
- `-DUTILISER_FPRINTF1 -DUTILISER_FPRINTF2 -DUTILISER_SORTIE_ERREUR2` en mettant un `\n` dans la première chaîne.
- ...

►Exercice 3. Fork :

Écrivez un programme qui affiche son pid sans revenir à la ligne, qui se duplique avec `fork(2)`, et dont l'enfant affiche son pid et celui de son parent. Le processus parent ne devra plus rien afficher après l'appel à `fork(2)`.

Proposez trois solutions à ce problème, et implémentez-en une qui vous paraît la plus pratique et/ou la plus élégante.

► **Exercice 4.** *Fork et redirection :*

Le but de cet exercice est d'écrire un programme `lance` qui s'exécutera comme suit :

`lance sortie commande arg1 arg2 ...`

1. *Écrivez un programme qui affiche la commande à exécuter sur sortie, puis exécute la commande à l'aide de la fonction `execvp`.*
2. *Étendez ce programme pour que le programme d'origine crée un nouveau processus qui exécutera la commande.*
3. *Dupliquez la sortie pour redigier la sortie de la commande exécutée sur la sortie passée en paramètre.*
4. *Une fois la commande `commande` exécutée, votre programme doit afficher des informations sur le statut de terminaison de `commande` (c.f. `wait(2)`). Écrivez un programme qui génère une faute de bus (`segfault`) et utilisez le pour tester le programme `lance`.*