Université de Strasbourg
Master CSMI – Semestre 2

Systèmes d'Exploitation
2020–2021

Contrôle sur table

Épreuve convoquée du 5 mai 2021
Durée : 1h30, tous documents papiers autorisés
Le barème est donné à titre indicatif

On souhaite rédiger le programme `xargexec` qui exécute des commandes dont les arguments sont des fichiers lus depuis l'entrée standard. La syntaxe est la suivante :

```
xargexec cmd [arg1 arg2 ...]
```

Ce programme exécute la commande `cmd` zéro, une ou plusieurs fois avec les arguments (optionnels) $\texttt{arg}_1$, $\texttt{arg}_2$, etc. suivis par les lignes lues depuis l'entrée standard. Par conséquent, le nombre d'exécutions de `cmd` correspond au nombre de lignes lues sur l'entrée standard. Par exemple, si le fichier `toto` contient 4 lignes :

```
turing> cat toto
readme.txt
exo18.c
test18.sh
sujet.pdf
```

alors la commande suivante exécutera 4 fois la commande `wc` avec l'option `-l` (1 fois par ligne présente dans le fichier `toto`) :

```
turing> ./xargexec wc -l < toto
378 readme.txt          # nombre de lignes du fichier readme.txt
666 exo18.c             # nombre de lignes du fichier exo18.c
223 test18.sh           # nombre de lignes du fichier test18.sh
42 sujet.pdf            # nombre de lignes du fichier sujet.pdf
```

Les différentes instances de `cmd` doivent s'exécuter en parallèle. Si toutes les commandes se sont exécutées sans erreur le programme se termine avec un code de retour nul, ou non nul dans le cas contraire.

Si un appel à `exec*` échoue, les appels suivants sont annulés pour éviter de créer inutilement des processus fils et le processus principal doit afficher sur la sortie d'erreur la raison de cet échec. Vous pourrez implémenter ce mode opératoire à l'aide d'un tube anonyme entre le processus père et un processus fils : si `exec*` échoue, alors le processus fils écrit la raison de l'erreur sur le tube. Lorsque le père lit n octets sur le tube avec $n > 0$ alors il sait que `exec*` a échoué et annule la création des processus fils suivants. Si $n = 0$, alors `exec*` a réussi et le processus père peut créer le processus fils suivant.

Vous pouvez utiliser des fonctions de bibliothèque pour l'affichage, les chaînes de caractères ainsi que pour l'allocation dynamique et la libération de mémoire. Il est demandé un contrôle strict des erreurs pouvant se produire, mais vous pouvez en faire une esquisse simple en utilisant la macro `CHK` et/ou la fonction `raler` qu'on supposera déjà écrites. L'inclusion des fichiers `.h` nécessaires est également supposée déjà effectuée. De même, vous pouvez utiliser une fonction `char *lire_ligne (void)` pour lire une ligne sur l'entrée standard et retourner un pointeur sur le buffer contenant la ligne lue. Le buffer ne contient pas de caractère de saut de ligne et se termine par le caractère `'\0'`. S'il n'y plus rien à lire sur l'entrée standard, cette fonction retourne `NULL`.

Question 1 – 3 points

Écrivez la fonction `char **creer_varg (int taille, char *vecteur [], char *ligne)`

qui construit un nouveau tableau de pointeurs dont le contenu est une copie des pointeurs présents dans `vecteur`, l'avant-dernier élément est une copie du pointeur `ligne`, et le dernier élément vaut `NULL`. Le nombre d'éléments dans `vecteur` est égal à `taille`. Cette fonction retourne un pointeur vers le nouveau tableau ainsi créé.

Question 2 – 4 points

Écrivez la fonction `void executer (char *arguments [], int tube_w)`

qui appelle la fonction `execvp` pour exécuter le programme indiqué avec ses arguments dans le tableau `arguments`. Si `execvp` échoue, la raison de l'échec doit être écrite sur le descripteur `tube_w` (qui référence le côté écriture d'un tube) avant de terminer le processus courant avec un code de retour non nul. Cette fonction ne retourne jamais. Vous pouvez convertir le numéro d'une erreur en chaîne de caractères grâce à la fonction `char *strerror (int errnum)`.

Question 3 – 3 points

Écrivez la fonction `void lancer_fils (char *vecteur [], int tube [])`

qui crée un processus fils. Après sa création, le processus fils appelle la fonction `executer` avec les bons arguments. Le paramètre `vecteur` contient le nom de la commande à exécuter ainsi que les arguments et se termine par `NULL`. Le paramètre `tube` correspond à un tube anonyme déjà créé.

Question 4 – 3 points

Écrivez la fonction `int attendre_fils (int n)`

qui attend la terminaison de n processus fils. La fonction retourne une valeur nulle si tous les fils ont terminé via un appel à `exit(0)` ou non nul dans le cas contraire.

Question 5 – 4 points

Écrivez la fonction `int traiter_une_ligne (char *ligne, int argc, char *argv [])`

qui prend la ligne lue sur l'entrée standard et les arguments de la fonction `main`, crée le tube ainsi que le tableau d'arguments avec `creer_varg` avant d'appeler `lancer_fils`. Cette fonction renvoie 1 si tout s'est bien passé. Dans le cas où `execvp` échoue dans un processus fils, cette fonction récupère la raison de l'erreur (sous la forme d'une chaîne de caractères) depuis le tube, l'affiche sur la sortie d'erreur et retourne 0.

Question 6 – 3 points

Écrivez la fonction `int main (int argc, char * argv [])`

Le programme prend en argument le nom de la commande à exécuter ainsi que des arguments optionnels. Pour chaque ligne lue sur l'entrée standard, cette fonction appelle `traiter_une_ligne`. Lorsqu'il n'y a plus rien à lire sur l'entrée standard, le processus père attend la terminaison des éventuels processus fils avec `attendre_fils`. Si `traiter_une_ligne` retourne 0 alors il faut directement appeler `attendre_fils`. Si toutes les commandes se sont exécutées sans erreur, cette fonction se termine avec un code de retour nul, ou non nul dans le cas contraire.