Université de Strasbourg
Master CSMI – Semestre 2

Systèmes d'Exploitation
2021–2022

Contrôle sur table

Épreuve convoquée du 12 mai 2022
Durée : 1h30, tous documents papiers autorisés
Le barême est donné à titre indicatif

On souhaite rédiger un programme ayant la syntaxe suivante :

`ronde` *`fichier`* v_0 v_1 `...` v_{n-1}

Ce programme est composé d'un processus père et de n processus fils p_i (avec $0 \leq i < n$). Le père communique la valeur v_i à chaque p_i qui doit la transmettre à son voisin, c'est-à-dire au processus p_k où $k = (i+1) \bmod n$. Comme utiliser des tubes serait trop simple, on utilise un fichier pour communiquer les valeurs. Cependant, le processus p_k ne doit pas chercher à lire une valeur dans le fichier tant qu'il n'est pas sûr que le processus p_i ait écrit cette valeur. Pour avoir cette assurance, il faut recourir à l'API POSIX des signaux.

On propose donc le schéma [1] suivant en plusieurs étapes :

1. le processus père génère les processus fils p_i (en leur communiquant i, la valeur v_i à transmettre et le nom du fichier) et écrit les identifiants des processus à la suite dans le fichier ;
2. lorsque tous les identifiants sont dans le fichier, le père envoie le signal `SIGUSR1` à chaque fils ;
3. dès réception de `SIGUSR1`, p_i peut alors lire le k-ème identifiant de processus dans le fichier ;
4. ce k-ème emplacement dans le fichier peut alors servir au processus p_i qui y écrit la valeur v_i ;
5. le processus p_i envoie alors le signal `SIGUSR2` au processus p_k pour lui indiquer que la valeur v_i est disponible dans le k-ème emplacement du fichier ;
6. dès réception de `SIGUSR2`, le processus p_k peut lire la valeur v_i dans le k-ème emplacement ; il la lit, puis l'écrit sur la sortie standard avant de se terminer ;
7. le processus père attend tous les processus, puis il supprime le fichier et se termine avec un code de retour nul si tous les fils se sont terminés sans erreur, ou non nul si au moins un fils a rencontré une erreur.

Vous utiliserez uniquement les primitives systèmes et non les fonctions de bibliothèque, sauf pour les affichages ou indication contraire dans l'énoncé. Il est demandé un contrôle strict des erreurs pouvant se produire, mais vous pouvez en faire une esquisse simple en vous aidant de la classique macro `CHK` et/ou de la toute aussi classique fonction `void raler(int, char *, ...)`.

Les questions peuvent être traitées indépendamment. Lorsqu'une question nécessite l'utilisation de fonctions demandées dans des questions antérieures, celles-ci seront supposées déjà écrites. Il est toutefois conseillé de lire l'énoncé jusqu'au bout avant d'entamer la rédaction.

Question 1 – 2 pts

Écrivez la fonction :

```
void args (int argc, const char *argv [], const char **fich, int tv [], int *pn)
```

qui vérifie les arguments reçus par `main` suivant la syntaxe donnée en introduction. Cette fonction renvoie le nom du fichier à l'adresse pointée par `fich` (sans recopier la chaîne de caractères) et les valeurs v_i dans le tableau `tv`. La taille de ce tableau est indiquée en entrée par l'entier pointé par `pn`, qui doit valoir en sortie le nombre n de valeurs trouvées. Si la syntaxe est invalide, ou si n est nul ou supérieur à la taille du tableau `tv`, une erreur doit être affichée. Vous pouvez utiliser la fonction `atoi` dans cette question.

1. Ce schéma suppose que la taille (`sizeof`) d'une valeur v_i (type `int`) soit inférieure ou égale à la taille d'un type `pid_t`. On supposera pour simplifier que ces tailles sont identiques, ce qui est le cas en pratique avec Linux sur les architectures de type Intel.

Question 2 – 3 pts

On s'intéresse maintenant à la manière dont les processus fils p_i vont recevoir les signaux `SIGUSR1` et `SIGUSR2` : on souhaite avoir deux variables globales `recu1` et `recu2` qui valent 0 tant que le signal correspondant n'a pas été reçu, puis 1 lorsque le signal est reçu.

Écrivez la déclaration de ces deux variables, la ou les fonctions appelées lors de la réception de ces signaux, et la fonction `void preparer_signaux (void)` qui prépare un processus pour la réception de ces signaux.

Question 3 – 2 pts

Écrivez la fonction `void attendre_signal (int signum)` qui attend la réception du signal indiqué (`SIGUSR1` ou `SIGUSR2`). Vous attacherez un soin particulier à ne pas risquer d'attendre éternellement le signal souhaité.

Question 4 – 5 pts

Écrivez la fonction `void fils (const char *fichier, int i, int vi)` appelée par le père. Elle réalise les étapes 3 à 6 du schéma décrit en introduction. Vous déterminerez le nombre n de processus fils à partir de la taille du fichier.

Question 5 – 4 pts

Écrivez la fonction `void lancer (const char *fichier, const int tv [], int n)` appelée par le père pour les étapes 1 et 2 du schéma décrit en introduction : cette fonction lance les n processus fils et écrit leur identifiant dans le fichier indiqué, puis leur envoie le signal de départ (`SIGUSR1`).

Question 6 – 2 pts

Écrivez enfin la fonction `main` suivant la syntaxe décrite en introduction. Vous n'oublierez pas de réaliser l'étape 7.

Question 7 – 2 pts

Où doit être appelée la fonction `preparer_signaux` et pourquoi ?