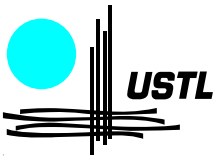

Licence d'informatique
Module de Programmation des systèmes

Examen première session 2008

Philippe MARQUET

Mai 2008

Durée : 3 heures.
Documents de cours, TD, et TP autorisés.
Les réponses seront concises et concrètes.

1 Réveil par `alarm`

La primitive

```
int sleep(unsigned seconds);
```

suspend l'exécution du processus courant pour `seconds` secondes.

Cette primitive `sleep()` était originellement implémentée à l'aide d'un minuteur géré par la primitive `alarm()` et de la délivrance du signal `SIGALRM`.

Exercice 1
Donnez une implémentation de `sleep()` basée sur `alarm()` et `SIGALRM`. On ne se préoccupera pas de la valeur retournée par `sleep()`. □

Exercice 2
Expliquez les inconvénients qui ont pu motiver l'abandon de ce type d'implémentation. □

2 Lister les fichiers controuvés

Il s'agit de développer une commande `controuve` dont l'objet est de lister les entrées sous le répertoire courant qui sont des liens symboliques référençant des fichiers n'existant pas dans le système de fichiers. On qualifie ces fichiers de controuvés.

On développe une fonction utilitaire préliminaire pour aider à l'implantation de cette commande.

Exercice 3
Donnez le code d'une fonction

```
int controuve_file(const char *pathname, const struct stat *st);
```

qui, considérant le fichier désigné par les deux paramètres `pathname` (son nom), et `st` (la valeur de type `struct stat` correspondante), teste si l'entrée dans le système de fichier est un fichier controuvé, c'est-à-dire :

- est un lien symbolique ;
- ce lien symbolique référence un nom de fichier non existant dans le système de fichiers.

La fonction `controuve_file()` retourne un booléen : une valeur `TRUE` si l'entrée est un fichier controuvé, une valeur `FALSE` sinon. □

La commande `controuve` va devoir parcourir le répertoire courant et ses sous-répertoires pour identifier les fichiers controuvés. Typiquement, les fonctions de parcours de la hiérarchie des répertoires telle notre commande `controuve` peuvent ou non suivre les liens symboliques rencontrés.

Exercice 4
Précisez quel doit être selon vous le comportement de la fonction `controuve` si le fichier désigné par le paramètre est un lien symbolique. □

Pour compléter le développement de la commande `controuve`, il reste maintenant à définir une fonction

```
void controuve(const char *pathname);
```

qui
- si l'entrée de nom `pathname` est un fichier controuvé, affiche le nom cette entrée ;
- traite les liens symboliques comme défini à la question précédente ;
- dans le cas d'un répertoire, vérifie pour chacune de ses entrées, et ce récursivement, s'il agit d'un fichier controuvé.

Dans ses grandes lignes, la fonction `main()` de la commande `controuve` consisterait en un simple appel à `controuve(".")`.

Exercice 5
Donnez le code de cette fonction

```
void controuve(const char *pathname);
```

□

3 Word count parallèle

La commande Unix `wc` compte les lignes, mots, et lettres des fichiers dont les noms lui sont fournis sur la ligne de commande :

```
% wc xam08*.c
      93     227    2031 xam08controuv.c
     154     410    3050 xam08rdv.c
      55     105    1014 xam08sleep.c
      45      86     723 xam08wc-ps.c
      63     126     975 xam08wc-th.c
     410     954    7793 total
```

On remarque que la commande affiche aussi les nombres de lignes, mots et caractères totaux. Dans le cas ou un unique paramètre est fourni à la commande, cette ligne de totaux n'est pas produite :

```
% wc xam08sleep.c
      55     105    1014 xam08sleep.c
```

Nous désirons implémenter une version parallèle de cette commande. Le principe est de déléguer le comptage de différents fichiers à différents fils d'exécution.

Deux approches d'une telle implémentation parallèle peuvent être imaginées : une approche à base de processus et une version à base de processus légers.

Exercice 6
Supposant disponible la commande Unix `wc`, donnez une implémentation d'une commande `wc-ps` qui crée autant de processus qu'il y a de fichiers sur la ligne de commande, chacun des processus prenant en charge le comptage d'un fichier. On ne se soucie pas d'afficher la dernière ligne donnant les totaux ; on ne se soucie pas non plus de l'ordre d'affichage des lignes du résultat. □

Exercice 7
Discutez de la possibilité de modifier votre commande `wc-ps` pour afficher, en fin d'exécution, la ligne fournissant les totaux. □

On suppose maintenant l'existence d'une fonction

```
struct wc_s {
    unsigned wc_w;              /* word */
    unsigned wc_l;              /* line */
    unsigned wc_c;              /* char */
};

int wc_file(const char *filename, struct wc_s *pwc);
```

qui retourne dans la structure pointée par `pwc` le nombre de mots, lignes et caractères du fichier `filename`. Cette fonction retourne une valeur non nulle en cas d'erreur.

On suppose aussi l'existence d'une fonction

```
int pr_wc_line(const char *str, const struct wc_s *pwc);
```

qui affiche sur la sortie standard une ligne de résultat comme le fait la commande `wc` pour chacun des fichiers ou pour l'éventuelle ligne des totaux.

Exercice 8
Donnez une implémentation d'une commande `wc-th` qui crée autant de processus légers qu'il y a de fichiers sur la ligne de commande, chacun des processus légers prenant en charge le comptage d'un fichier. On ne se soucie pas d'afficher la dernière ligne donnant les totaux. □

Exercice 9
Expliquez comment ajouter l'affichage par la commande `wc-th` de la ligne des totaux. □

Exercice 10
Donnez le code d'une implémentation de cette version étendue à l'affichage des totaux de la commande `wc-th`. □

4 Rendez-vous avec des tubes

Le principe d'un rendez-vous entre N processus est d'attendre que le N^e processus arrive au rendez-vous pour laisser continuer l'ensemble des processus.

Dans le cadre de cet examen, on propose une implémentation de rendez-vous entre processus à l'aide de tubes. Cette implémentation est basée sur la structure et le principe suivant :

- un processus *serveur* assure la gestion des rendez-vous entre N processus clients ;
- deux tubes sont utilisés : un tube entre le serveur et les clients et un tube entre les clients et le serveur ;
- la synchronisation est assurée par le fait que la lecture dans un tube vide est bloquante ;
- quand un client arrive au point de rendez-vous, il en informe le serveur en écrivant un caractère (mettons `'x'`) dans le tube ; il se met alors en attente du rendez-vous en lisant un caractère depuis le second tube ;
- le serveur effectue sans fin une boucle de gestion d'un rendez-vous. Cette gestion d'un rendez-vous consiste à attendre les N processus en lisant N fois le caractère `'x'` depuis le tube ; il libère alors les N processus en écrivant N caractères dans le second tube.

On utilisera la structure

```
struct rdv_s {
    unsigned rdv_n_process;     /* nombre de processus */
    int rdv_pipe_clt2srv[2];    /* tube clients vers serveur */
    int rdv_pipe_srv2clt[2];    /* tube serveur vers clients */
};
```

pour implémenter notre bibliothèque de gestion de rendez-vous.

Deux fonctions forment l'interface de cette bibliothèque : la première fonction

```
int rdv_init(struct rdv_s *rdv, unsigned n_process);
```

assure l'initialisation d'un mécanisme de rendez-vous pour `n_process` processus en mettant à jour la structure pointée par `rdv`. Cette initialisation consiste en la création des tubes et du processus serveur.

La seconde fonction

```
int rdv(const struct rdv_s *rdv);
```

est appelée par chacun des N processus et assure une synchronisation par rendez-vous.

L'utilisation de la bibliothèque nécessite l'initialisation de la structure `struct rdv_s` avant la création des processus clients.

Exercice 11
Expliquez comment la structure de données `struct rdv_s` est partagée entre les processus. □

Exercice 12
La fonction d'initialisation est en autre chargée de créer le processus serveur qui va exécuter la boucle sans fin de gestion des rendez-vous. Donnez le code de la fonction

```
static void rdv_server(const struct rdv_s *rdv);
```

qui sera exécutée par ce processus serveur. □

Exercice 13
Donnez le code de la fonction d'initialisation `rdv_init()` □

On se soucie maintenant de la libération des ressources associées à une structure `struct rdv_s` une fois les différents rendez-vous réalisés : il s'agit de fermer les descripteurs de fichiers associés aux deux tubes et de terminer le processus serveur.

Exercice 14
Proposez un mécanisme pour signaler au processus serveur que son travail est fini. Expliquez précisément le fonctionnement attendu. □

Fin du sujet