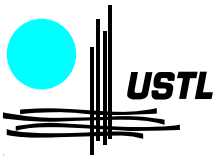

Licence d'informatique
Module de Programmation des systèmes

Examen seconde session 2005

Philippe MARQUET

Juin 2005

Durée : 3 heures.
Documents de cours et TD autorisés.

1 Tubes liant des fils

Soit le fichier `fils.c` suivant

```
int
main(int argc, char *argv[])
{
    int i;
    int fds[2];
    int in = STDIN_FILENO;
    int out = STDOUT_FILENO;

    for (i=0; i<argc-1 ; i++) {
        pipe(fds);
        if(fork() == 0) {
            close(fds[1]);
            in = fds[0];
        } else {
            close(fds[0]);
            out = fds[1];
            break;
        }
    }

    if (i!=0)
        printf("Processus %d: %s\n", i, argv[i]);
    else
        printf("Processus %d, %d fils\n", i, argc-1);

    exit(EXIT_SUCCESS);
}
```

qui sera compilé en un exécutable `fils`. On notera particulièrement que l'instruction `break` provoque la sortie de la boucle `for`.

Exercice 1
Donnez un schéma identifiant les processus et les tubes créés par l'invocation

```
% ./fils ls wc
```

sur ce schéma, vous figurerez, pour chacun des processus, les valeurs des variables `in`, `out`, et `i`.
Donnez le résultat affiché par cette exécution. □

On désire transformer ce programme pour que chacun des arguments de `fils` soit compris comme une commande qui sera exécutée par un processus ; les entrées et sorties standard des processus étant connectées entre elles. Ainsi, l'exécution de la commande

```
% ./fils ls wc
```

sera grossièrement équivalente à

```
% ls | wc &
```

Exercice 2
Complétez le code du programme `fils.c` pour ce faire. □

On désire de plus que l'exécution de la commande `fils` ne se termine pas avant que les processus fils aient tous terminés.

Exercice 3
Complétez à nouveau votre programme `fils.c` pour attendre cette terminaison des fils. □

2 Quelle synchronisation

Soit la fonction `fsync()` suivante

```
#define N    12

void
fsync()
{
    static pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
    static pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
    static int already = 0;

    pthread_mutex_lock(&lock);
    already++;
    if (already != N)
        pthread_cond_wait(&cond, &lock);
    already--;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&lock);
}
```

Exercice 4
Quelle synchronisation est réalisée par un groupe de processus légers appelant concurremment la fonction `fsync()` ? □

Exercice 5
Il est habituel d'encadrer les appels à la fonction `pthread_cond_wait()` par une boucle `while`. Ce n'est pas le cas dans cette fonction `fsync()`. Expliquez pourquoi ce n'est pas utile ici. □

Exercice 6
Soit le code suivant

```
void
sync()
{
    fa();
```

```
        fsync();
        fb();
        fsync();
        fc();
    }
```

appelé concurremment par un ensemble de processus légers. Expliquez comment les deux appels à la fonction `fsync()` peuvent interférer. □

Exercice 7
Comment éviter cette interférence lors de deux synchronisations successives ? Donnez une version corrigée de la fonction `fsync()`. □

Exercice 8
Expliquez comment généraliser cette solution à de multiples synchronisations successives par la fonction `fsync()`. □

3 Tubes à accès direct

Il a été remarqué que les opérations de positionnement (par exemple avec `lseek()`) ne peuvent être utilisées sur les tubes. Pour contourner cette contrainte, au moins en lecture, la proposition d'une fonction

```
int make_skfifo(int fd_fifo);
```

a été faite. Cette fonction retourne un descripteur sur un fichier temporaire qui a été créé et rempli avec le contenu lu sur le tube de descripteur `fd_fifo`. Avant de retourner le descripteur, la position courante est remise au début du fichier.

Exercice 9
Donnez le principe et le code C d'une implantation possible de la fonction `make_skfifo()`. On pourra reposer la création d'un nom de fichier temporaire sur la fonction de la bibliothèque C standard

```
#include <stdio.h>

char *tempnam(const char *dir, const char *pfx);
```

dont la description est fournie en annexe. □

On désire utiliser cette fonction `make_skfifo()` pour l'implantation d'une commande `prt` qui affiche sur la sortie standard le contenu d'un fichier précédé d'un entête. Cet entête indique, entre autres, le nombre de lignes du fichier. Le contenu de ce fichier est fourni sur l'entrée standard.

Dans le cas ou l'entrée standard correspond à un tube, la fonction `make_skfifo()` est utilisée. On se référera à l'annexe pour la description de la structure `structstat`) permettant de déterminer si un fichier est un tube.

On utilisera la fonction

```
int prt_header(int nlines);
```

qui affiche sur la sortie standard un entête avec comme information de nombre de lignes la valeur fournie par `nlines`.

Exercice 10
Donnez le principe et le code C d'une implantation de la commande `prt`. □

Si la fonction `make_skfifo()` peut être utile dans certaines circonstances, son champ d'application n'est pas universel. Il est des situations où l'utilisation de la fonction est soit impossible, soit amène à une implantation inefficace.

Exercice 11
Identifiez des situations pour lesquelles l'utilisation de la fonction `make_skfifo()` n'est pas adaptée. □

Le fichier temporaire crée par la fonction `make_skfifo()` doit être détruit. Une proposition est d'assurer cette destruction *automatiquement* lorsque toutes les références au fichier seront fermées. La proposition est d'ajouter dans l'implantation de `make_skfifo()` un appel à `unlink()` avant de retourner le descripteur.

Exercice 12
Expliquez le mécanisme qui garantit que cette proposition respecte cette destruction « automatique » du fichier temporaire. □

A Éléments d'aide

Cette annexe comporte la description de quelques fonctions pouvant être utilisées dans vos réponses.

Fonction `tempnam()`

La fonction de la bibliothèque C standard `tempnam()` crée un nom qui pourra être utilisé pour un fichier temporaire :

```
#include <stdio.h>

char *tempnam(const char *dir, const char *pfx);
```

L'utilisateur choisit le répertoire dans lequel le fichier devra être créé, argument `dir`. L'utilisateur peut aussi choisir un préfixe au nom du fichier. L'argument `pfx`, s'il est non nul, doit référencer une chaîne d'au plus cinq caractères qui sera utilisée comme préfixe.

En cas de succès, `tempnam()` alloue dynamiquement la mémoire nécessaire au résultat, y copie le nom absolu du fichier et retourne un pointeur sur cette mémoire. Ce pointeur devrait être passé en argument à `free()` par l'utilisateur pour libérer cette mémoire.

En cas d'erreur, un pointeur nul est retourné et `errno` et positionné.

Par exemple, l'extrait de code

```
char *fname;
fname = tempnam("/tmp", "foo");
if (fname)
    printf("%s\n", fname);
```

pourrait afficher le résultat suivant :

```
/tmp/foo0029ab
```

Attention, la fonction `tempnam()` crée juste le nom de fichier. C'est à l'utilisateur de créer et de détruire le fichier.

Structure `structstat`

La structure `structstat` retournée par les primitives `fstat()` et `lstat()` et `stat()` comporte les champs suivants :

```
struct stat {
    dev_t     st_dev;
    ino_t     st_ino;
    nlink_t   st_nlink;
    mode_t    st_mode;
```

```
uid_t     st_uid;
gid_t     st_gid;
off_t     st_size;
blkcnt_t  st_blocks;
...
```

parmi lesquels le champ `st_mode` est utilisé pour identifier le type du fichier. Les macros suivantes peuvent être utilisées sur ce champ :

`S_ISDIR(m)` teste si le fichier est un répertoire.

`S_ISFIFO(m)` teste si le fichier est un tube.

`S_ISREG(m)` teste si le fichier est régulier.

`S_ISLNK(m)` teste si le fichier est un lien symbolique.

`S_ISSOCK(m)` teste si le fichier est une prise (socket).

Fin du sujet