Licence d'informatique
Module de Programmation des systèmes

Examen première session 2005

Philippe MARQUET

Mai 2005

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

1 Fichiers temporaires

L'objectif est la construction d'une fonction `mktmpfile()` de création d'un fichier temporaire à la manière de la fonction `tmpfile()` existante dans la bibliothèque standard. Un tel fichier temporaire sera unique, et ainsi le programme l'ayant créé pourra en faire librement usage.

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. Entre le moment de création du nom et le moment auquel le fichier est créé, il est possible qu'un autre processus ait créé un fichier avec ce même nom.

Exercice 1
Quel mode d'ouverture de fichier permet d'identifier qu'un autre processus n'a pu créer le fichier entre temps ? □

On notera que dans ce cas, la primitive d'ouverture de fichier retourne une erreur (-1) et positionne `errno` à `EEXIST` si le fichier existe déjà.

La fonction

```
int mktmpfile(void);
```

retourne un descripteur sur un fichier temporaire qui vient d'être créé et ouvert en lecture/écriture. Cette fonction fait des appels à la fonction `tempnam()` jusqu'à ce que le fichier dont le nom est retourné puisse être ouvert en création par le programme sans erreur. Ce fichier temporaire est créé dans le répertoire désigné par la variable d'environnement `$TMPDIR` si elle est positionnée, dans le répertoire `/tmp` sinon. En cas d'erreur (par exemple allocation mémoire impossible...) la fonction retourne la valeur -1.

Exercice 2
Donnez le principe et le code C d'une implantation de la fonction `mktmpfile()`. □

La documentation de la fonction standard `tmpfile()` indique que le fichier temporaire créé sera *automatiquement* détruit lorsque toutes les références au fichier seront fermées. L'implantation habituelle de `tmpfile()` assure cette fonctionnalité par un appel à `unlink()` avant de retourner le descripteur.

Exercice 3
Expliquez le mécanisme qui garantit que l'implantation habituelle respecte cette destruction « automatique » du fichier temporaire. □

Exercice 4
Une solution pour adresser le problème de l'éventuelle création du fichier temporaire par un autre processus serait de regrouper l'appel à la fonction `tempnam()` et l'ouverture du fichier dans une construction identifiant une section critique. Expliquez pourquoi cette solution n'est cependant pas envisageable. □

2 Nouvelle implantation de la commande `cp`

Il s'agit de définir une nouvelle implantation, plus efficace, de la commande `cp` de copie de fichier. On se limite à la copie d'un fichier source unique dans un fichier destination.

La version 2.4 du noyau Linux introduit, comme d'autres systèmes, un appel système non standard `sendfile()` pour réaliser efficacement la copie d'un fichier. Le prototype de cet appel système est le suivant :

```
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

Cet appel système copie des données entre un descripteur de fichier et un autre. Le descripteur `in_fd` doit être ouvert en lecture, et `out_fd` en écriture. Le pointeur `offset` doit correspondre à une variable contenant la position dans le fichier d'entrée à partir de laquelle `sendfile()` commencera la lecture. Lorsque la routine se termine, cette variable pointée par `offset` est remplie avec la position de l'octet immédiatement après le dernier octet lu. L'argument `count` est le nombre d'octets à copier entre les descripteurs de fichiers. La fonction retourne le nombre d'octets écrits, -1 en cas d'erreur.

Exercice 5
Donnez le principe et le code C d'une implantation d'une fonction `copy_file()` à l'aide de `sendfile()` ; la fonction

```
int copy_file(const char *src, const char *dst);
```

copiant d'un fichier source de nom `src` dans un fichier destination de nom `dst`. □

L'introduction de l'appel système `sendfile()` a été motivée par une double considération :
- limiter le coût de transferts mémoire lors de la réalisation de copies de fichiers ;
- limiter le nombre de changements de mode du processeur lors de la réalisation de copies de fichiers.

Exercice 6
Pour chacun des deux points : transfert mémoire, et changement de mode du processeur, explicitez en quoi l'introduction de l'appel système `sendfile()` rend plus efficace la réalisation de copie de fichiers. □

3 États des processus

Parmi les états d'un processus, nous avons identifié :
- actif (prêt ou élu) ;
- bloqué ;
- stoppé (aussi nommé suspendu).

Il s'agit ici de mettre en évidence ces états des processus.

Exercice 7
Utilisez l'annexe jointe au sujet pour lister les événements qui sont associés aux six transitions entre ces trois états. Certaines transitions ne sont pas possibles ; dans ce cas expliquez pourquoi. □

Exercice 8
Pour chacun de ces trois états, donnez le principe et le code C permettant la création d'un processus dans cet état. □

4 Quelle synchronisation

Soit la fonction `f()` suivante

```
#define N    12

void
f()
{
    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);
    else
        already = 0;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&lock);
}
```

Exercice 9
Quelle synchronisation est réalisée par un groupe de processus légers appelant concurremment la fonction `f()` ? □

Exercice 10
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 `f()`. Expliquez. □

Exercice 11
Quel problème peut survenir pour un groupe des processus légers voulant réaliser de multiples synchronisations successives par la fonction `f()` ? □

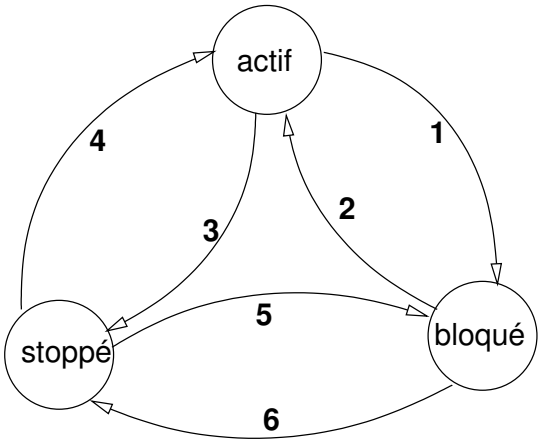
Exercice 12
Comment corriger ce problème ? Donnez une version corrigée de la fonction `f()`. □

Fin du sujet

Épreuve de **Programamtion des systèmes**
Licence d'informatique

Numéro de place : .....
Intercalaire : ..... / .....

Exercice 7



Transition 1 de actif à bloqué

Transition 2 de bloqué à actif

Transition 3 de actif à stoppé

Transition 4 de stoppé à actif

Transition 5 de stoppé à bloqué

Transition 6 de bloqué à stoppé