Examen de Programmation Systèmes - Master I
Jeudi 19 juin - Durée : 3 heures

Exercice 2.
On considère le programme suivant :

```
/* binaire : prog */
#include <stdio.h>
main(int argc, char*argv[]) {
int i;
if (argc == 1) return;
printf("%d,%d --> %d : ", getpid(), getppid(), argc);
for(i = 1; i < argc; i++) printf("%s ", argv[i]);
putchar('\n');
if(fork() == 0){
    argv[argc-1] = NULL;
    main(argc-1, argv);
    return;}
while (wait(NULL) != -1); /* ******** */
argv[argc-1] = NULL;
main(argc-1, argv);
return;
}
```

2.1. Que produit l'exécution de la commande

```
./prog a b c d
```

En particulier :
– quelle est la hiérarchie des processus créés (on supposera que le premier a comme numéro N et que les numéros des processus sont consécutifs) ?
– qu'est ce qui est affiché ?
2.2. Qu'est ce que cela change du point de vue de l'exécution si on supprime la ligne repérée par des `*` ?
2.3. Est-il possible d'obtenir un programme équivalent (même hiérarchie de processus et mêmes affichages) en remplaçant les appels à la fonction `main` par des appels à `exec`. Si oui donner le code correspondant et si non pouquoi n'est-ce pas possible ?

Exercice 3.
On se propose de réaliser un outil de compression parallélisé (*i.e.* qui effectue plusieurs tâches en même temps pour accélérer les calculs). On supposera que les fichiers à compresser seront systématiquement découpés en n tranches, chacune pouvant être compressée indépendamment des autres. Le résultat final étant un fichier contenant la concaténation ordonnée des différentes tranches compressées.
Pour réaliser ce compresseur, on utilisera n processus (autant que de tranches). On notera p_i $(i \in [1, n])$ le processus chargé de la tranche i. Les processus sont *chaînés* entre eux par des tubes. On notera t_i, $(i \in [1, n-1])$ le tube qui relie p_i à p_{i+1} et permet à p_i d'envoyer des données à p_{i+1}.
La compression d'un fichier f se passe de la façon suivante. Le processus p_i lit les données qui lui correspondent dans le fichier f (octets $[(i-1)\frac{|f|}{n}, i\frac{|f|}{n} - 1]$). La compression d'un segment sera réalisée à l'aide d'un appel à la fonction `extern int compresse(void *addresse,int l) ;` et qui compresse les `l` octets situés à l'`adresse` donnée en argument ; le résultat est aussi situé à l'`adresse` donnée et la longueur du segment après compression est renvoyé en retour. **Attention :** cette fonction vous est donnée comme existante, vous n'avez pas à l'écrire, juste à l'utiliser.

Lorsque la compression d'un segment est terminée, le processus p_i $(i > 1)$ attend de son prédécesseur p_{i-1} qu'il lui envoie, par l'intermédiaire du tube qui les unis, la position à laquelle il devra écrire son segment compressé ; une fois l'écriture réalisée, il enverra au processus suivant la position à laquelle celui-ci devra écrire ses données, etc. On trouvera dans la figure 1, un exemple illustrant la compression d'un fichier réalisée à l'aide de 7 processus (7 segments).

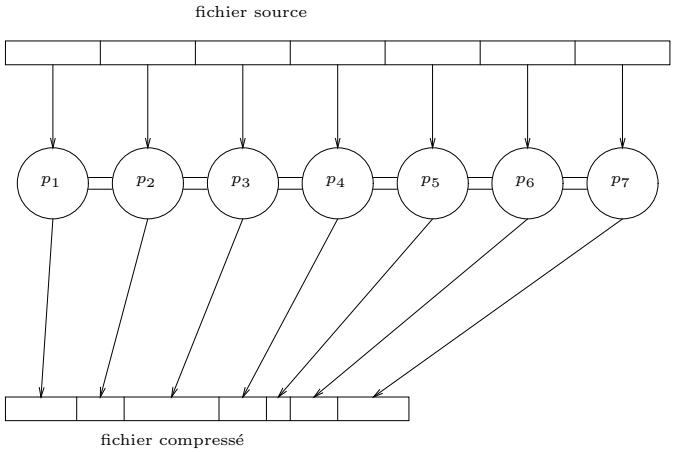

FIG. 1 – Un exemple de compression utilisant 7 processus.

La commande de compression devra pouvoir être lancée à l'aide de la commande suivante :
`comprime` [`-`n] $fichier$
où n (1 par défaut) désigne le nombre de processus qui seront employés pour réaliser la compression et $fichier$ le nom du fichier à compresser.
3.1. Décrire succinctement l'architecture employée, la filiation des processus, les messages echangés, etc.
3.2. Écrire en langage C, le code de la commande en question. On prendra soin des terminaisons possibles (normales ou non).

Dans cette partie, la compression de n segments sera réalisée par p processus (simulation d'un *pool* de processeurs). Dans ce cas, l'idée générale est la même, sauf qu'on considérera le cas particulier où $p < n$ et qu'il faudra donc faire en sorte que lorsqu'un segment a été écrit, le processus se charge alors de compresser un nouveau segment adéquat, *i.e.* le processus p_i compressera successivement tous les segments (s'ils existent) $[(i-1+kp)\frac{|f|}{n}, (i+kp)\frac{|f|}{n}-1]$ pour $k \geq 0$. La figure 2 illustre le mécanisme employé dans le cas de 7 segments compressés par 3 processus.
La commande de compression devra pouvoir être lancée à l'aide de la commande suivante :
`comprime2 -`p [`-`n] $fichier$
où p désigne le nombre de processus qui seront employés pour réaliser la compression, n (1 par défaut) le nombre de segments du $fichier$ à compresser.
3.3. Décrire succinctement l'architecture employée, la filiation des processus, les messages echangés, etc.
3.4. Écrire en langage C, le code de la commande en question. On prendra soin des terminaisons possibles (normales ou non).

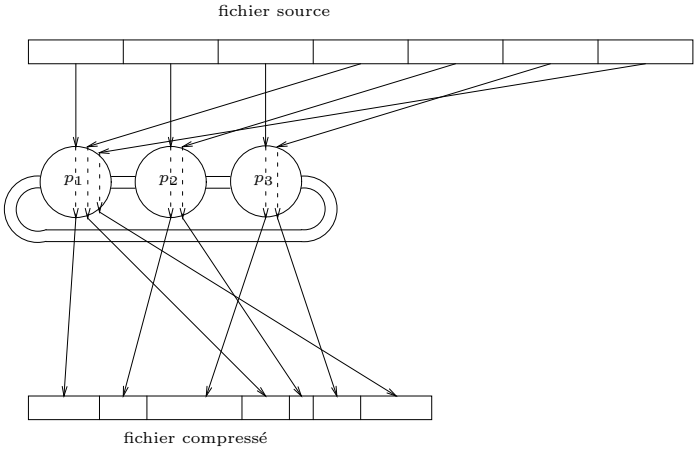

FIG. 2 – Un exemple de compression de 7 segments à l'aide de 3 processus.