```
    if (child1==0) child2=fork();
  } while (child1==0 && n<4);
  wait(&status);
  return EXIT_SUCCESS;
}
```

1. *Combien de processus sont exécutés en tout ? Quelles sont les valeurs de* `n` *dans les différents processus ? Vous justifierez votre réponse.*

2. *La variable* `n` *a-t-elle la même adresse pour tous les processus ? Est-elle partagée ?*

▸**Exercice 4.** *Communication inter-processus*

On souhaite réaliser une communication d'un tableau d'entiers entre deux processus. Le premier processus écrit les éléments du tableau puis envoie le signal SIGUSR1 au second. Le second fait la somme de tous les éléments du tableau et met 0 comme valeur du premier élément de tableau pour informer le premier qu'il a fini la lecture. La communication du tableau se fait par un fichier partagé, nommé `file.txt`*, en utilisant la fonction* `mmap`*.*

Ecrire le programme du deuxième processus, celui qui attend, fait la somme des éléments et met le premier à 0. On ne fera aucune gestion des erreurs et on omettra les includes. Attention, votre code ne doit pas excéder 15 lignes !

▸**Exercice 5.** *Threads et synchronisation*

On considère un programme simulant le comportement d'un distributeur de billets. Une fois que l'utilisateur s'est identifié et a indiqué le montant demandé, le distributeur exécute le programme suivant:

```
void retire(int compte, int demande, int date) {
  if (Banque[compte].solde >= demande) {
    Banque[compte].solde = Banque[compte].solde - demande;
    Banque[compte].dernier_retrait = demande;
    Banque[compte].date = date;
  }
}
```

avec `Banque` *un tableau global, partagé entre tous les distributeurs,* `compte` *est l'identifiant du compte de l'utilisateur,* `demande` *est le montant demandé. L'élément* `Banque[compte]` *comporte notamment plusieurs champs:*

- `solde`*, qui est la solde restant sur ce compte. La banque tient à ce qu'un utilisateur ne puisse pas prélever plus d'argent qu'il n'en a sur son compte...*
- `dernier_retrait`*, qui est le dernier montant prélevé,*
- `date`*, qui est la date du dernier retrait.*

Il y a plusieurs distributeurs d'argent associés à la même banque, et on les considère dans l'exercice comme des threads d'un même processus.

1. *Quelle est la section critique du code précédent et expliquer comment il est possible de retirer plus d'argent qu'il n'y en a sur un compte.*

2. *Réécrire la fonction* `retire` *pour garantir l'exclusion mutuelle sur la section critique, en utilisant un seul verrou. Utiliser les fonctions* `pthread_mutex_lock`*,* `pthread_mutex_unlock` *et on dira quelles variables globales doivent être déclarées et comment elles sont initialisées dans le* `main`*.*

3. *Quel est l'inconvenient de la solution proposée à la question précédente ? En supposant que vous pouvez ajouter un ou des champs à* `Banque[compte]`*, quelle solution permet de garantir l'exclusion mutuelle sans cet inconvénient ? Ecrire le nouveau code de* `retire`*. Ecrire de plus le code d'initialisation des nouveaux champs.*

4. *On considère maintenant une opération de consultation du compte, qui informe l'utilisateur du solde restant sur son compte ainsi que de la dernière opération et de sa date.*