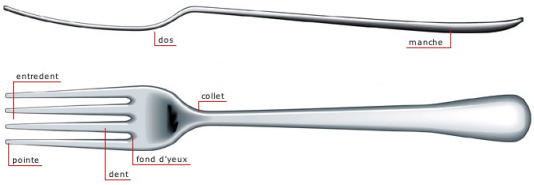
TP 2 - Création de processus - Fork



Le répertoire de ce TP dans votre dépôt s'appellera "TPFork".

Comme le nom du TP l'indique, vous allez utiliser la fonction système `fork` pour créer de nouveaux processus.

Attendu

Vous avez deux séances pour réaliser ce TP.

Fichiers attendus:

- `OS_Nom1_Nom2`
  - `TPFork`
    - `.gitignore` (avec tout ce qui n'est pas attendu)
    - `Makefile`
    - `reponses.md`
    - `forkWait.c`
    - `forkGlobales.c`
    - `forkShared.c`
    - `forkSharedWait.c`
    - `forkBonus.c`

Ce qui n'est pas poussé sur votre dépôt GitLab n'est pas corrigé.

Pensez donc bien à effectuer des `commit` et des `push` régulièrement.

Sujet

L'objectif de ce TP est de vous faire créer et manipuler des processus en C.

Un processus Unix peut en créer un autre grâce à l'appel système `fork`. Un processus qui appelle `fork` est **dupliqué** par le système.

Après cette **duplication** il existe un nouveau processus, appelé *processus enfant* qui est une copie exacte du processus initial qui a appelé `fork` et qui est appelé *processus parent*. L'exécution des deux processus continue en parallèle après l'appel à `fork`. On distingue le parent de l'enfant par la

valeur renvoyée par `fork`:

- dans le processus enfant : **0**
- dans le processus parent : le numéro d'identification (PID) du processus **enfant** créé

Pour ce TP, vous aurez besoin du document intitulé : "Unix : quelques fonctions système". Les informations données dans ce document sont une synthèse, vous pouvez toujours, pour chaque fonction Unix, lire le manuel en ligne correspondant avec :

```
man 2 <fonction>
man 3 <fonction>
```

Mise en place

Vous trouverez un exemple de création d'un processus dans le fichier `fork.c`

- Si ce n'est pas déjà fait, créez un répertoire TPFork dans votre dépôt.
- Enregistrez ce fichier dans le répertoire **TPFork**.
- Ajoutez vos informations, **noms** et **groupe de TP**,
- Commitez puis poussez le.

Compilation

Vous devez créer un `Makefile` permettant de compiler ce programme d'exemple puis les autres que vous réaliserez. Vous êtes **notés** sur ce `Makefile`, *poussez-le*.

1. Identification et synchronisations simples

Vous ferez les modifications demandées dans votre fichier `forkWait.c`. Vous répondrez aux questions de cette section dans un fichier `reponses.md`.

Un exemple de fichier `reponses.md` est à votre disposition.

Ajoutez également ce fichier à votre dépôt GitLab pour qu'il soit pris en compte.

```
git add responses.md
git commit
git push
```

1.1. Questions de filiation

Le programme fourni (`fork.c`) réalise pour chacun des processus, parent et enfant, l'affichage de

leur `PID` et du `PID` de leur parent (fonctions `getpid` et `getppid`). La durée de vie du parent a été rallongée pour permettre laisser au processus enfant le temps de s'exécuter avant la fin de son parent. Faites tourner ce programme.

1. Quel est le PID du processus parent ? Quel est le PPID du processus enfant ? Qu'est-ce que cela vous permet de vérifier ?
2. Quel processus est le parent du processus parent ? Retrouvez son nom et son PID avec:

   ```
   pstree -ps <numéro de processus>
   ```

3. Modifiez le programme pour que l'enfant réalise son affichage 4 fois d'affilé avec une pause de 1 seconde entre chaque affichage. Comment évoluent le `PID` et le `PPID` du processus enfant une fois que son parent ``naturel'' se termine ? Expliquez ce qui arrive à un processus orphelin et quel processus l'adopte.

1.2. Synchronisations

Vous passerez à un fichier ***forkSync.c*** pour cette question.

Afin de mettre en œuvre la synchronisation de processus, vous allez créer deux processus enfants puis attendre la fin de leur exécution et afficher de l'information relative à l'enfant qui se termine.

Commencez par faire en sorte que le processus parent ait deux enfants mais pas de petits-enfants. **Ceci n'est pas trivial**

Ajoutez des délais d'attente avant la fin des processus enfants.

Faites ensuite en sorte que le parent attende la fin de ses deux enfants et affiche leurs `PID`.

Vous aurez besoin de la fonction `wait`. Exécutée par un processus, elle bloque ce dernier jusqu'à la fin d'un de ses enfants. Si le processus n'a pas d'enfant actif, `wait` renvoie -1, dans le cas contraire, elle renvoie le numéro (`PID`) du processus enfant mort. Les enfants pourront se contenter d'afficher leur `PID`.

2. Variables Globales

Cette partie se fera dans un fichier nommé `forkGlobales.c`

En repartant du fichier `fork.c` d'origine, définissez une variable globale de type entier puis faites en sorte que chaque processus, le parent *et* l'enfant, incrémente puis affiche la valeur de cette variable un grand nombre de fois (au moins 100). Assurez vous à l'aide d'affichages que les deux processus accèdent à la variable globale de manière simultanée en vérifiant que l'un **ne se termine pas** avant que l'autre ne commence. Si ce n'est pas le cas, ajoutez des itérations d'incrément.

1. Y a-t-il interférence entre les deux processus ?

Répondez et **expliquez pourquoi** dans le fichier `reponses.md`, que *vous avez ajouté* à votre dépôt Git.

> Identifiez **clairement** la question à laquelle vous répondez dans le fichier `reponses.md`

Pour finir cette partie, initialisez la valeur de la variable globale à une valeur particulière (autre que 0) puis ajoutez un affichage permettant de vérifier que le fork duplique l'espace mémoire du processus parent **dans l'état dans lequel il est au moment du `fork`**.

3. Fichier Partagé

> Vous travaillerez dans un fichier nommé `forkShared.c` pour cette partie.

Afin d'expérimenter l'accès à des ressources partagées il vous est demandé de faire en sorte que deux processus accèdent simultanément à un fichier.

Pour cela écrivez un programme qui ouvre un fichier `sharedFile.txt` en écriture **puis** qui crée un processus enfant. Les deux processus devront ensuite écrire *simultanément* dans le fichier une chaîne de votre choix un grand nombre de fois (~1000 fois).

```
for (int i = 0; i < 1000; i++) {
   fprintf(sharedFile, "Je suis le pere !![%ld:%d]\n", pid, i);
   fflush(sharedFile);
}
```

Faites en sorte que la chaîne écrite par les deux processus soit *longue*. Par exemple `"Je suis le fiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiils"`. Ou plus long.

Vous ajouterez également un délai avant la première des écritures du parent dans le fichier pour donner au processus enfant le temps de démarrer avec : `usleep(10);` (Expérimentez avec la durée de ce délai).

Lancez plusieurs fois votre programme et observez, dans un éditeur de texte, le contenu du fichier `sharedFile.txt` après chaque lancement (pensez à recharger le fichier après une exécution de votre programme).

Vous devriez constater un mélange des affichages du parent et de l'enfant.

Vous répondrez aux questions suivantes dans le fichier `reponses.md`:

1. Expliquez pourquoi il y a un mélange des affichages. (Si ce n'est pas le cas, allongez les chaînes, modifiez le délai de démarrage de l'affichage du parent)

Pour l'ouverture du fichier, vous utiliserez la fonction `fopen` et le mode `"w"`.

```
FILE* sharedFile = fopen("./sharedFile.txt", "w");
```

Pour écrire dans le fichier, vous utiliserez la fonction `fprintf`.

4. Fichier Partagé et synchronisation

Vous travaillerez dans un fichier nommé `forkSharedWait.c` pour cette partie.

Afin d'éviter le mélange des affichages, vous allez ouvrir le fichier une fois les processus créés puis synchroniser les processus

Ouverture par chaque processus

Dans la question précédente, l'ouverture du fichier partagé se fait, si vous avez respecté les consignes, avant la création du processus enfant. Déplacez l'ouverture du fichier pour qu'elle ait lieu **après** la création de l'enfant, après le `fork` donc.

Synchronisation

Attendez que le processus enfant ait terminé son exécution avant de commencer les affichages du parent.

Vous prendrez soin de **numéroter les affichages** pour vérifier qu'ils ont tous lieu. Inspirez vous de l'exemple de code ci-dessus.

Vous répondrez aux questions suivantes dans le fichier `reponses.md`:

1. Constatez-vous toujours un mélange des affichages ?
2. Est-ce que les affichages des deux processus, notamment ceux du processus enfant, apparaissent dans le fichier ? Expliquez pourquoi.

Pour que les affichages du processus enfant apparaissent, utilisez le mode d'ouverture `"a"` **uniquement pour le parent**. Vérifiez que les affichages de l'enfant apparaissent bien dans le fichier partagé.

5. Bonus

Vous travaillerez dans un fichier nommé `forkBonus.c` pour cette partie.

Faites Exécuter la commande `ps -fU <votreLogin>` dans un processus enfant. Vous aurez besoin de la fonction `execlp`.

Ajoutez un affichage dans le processus enfant après le `execlp` pour indiquer la fin du programme `ps`.

1. Pourquoi cet affichage n'apparaît jamais ?

Ajoutez enfin un affichage dans le processus parent indiquant la fin de l'exécution de `ps`.

6. Compte-rendu

Pensez à vérifier que vous avez bien poussé votre dernière version des fichiers demandés.

Fichiers attendus, rappel:

- `OS_Nom1_Nom2`
  - `TPFork`
    - `.gitignore` (avec tout ce qui n'est pas attendu)
    - `Makefile`
    - `reponses.md`
    - `forkWait.c`
    - `forkGlobales.c`
    - `forkShared.c`
    - `forkSharedWait.c`
    - `forkBonus.c`

> Ce qui n'est pas poussé sur votre dépôt GitLab n'est pas corrigé.