R3.05 - TP 6
Tubes (suite 2)

Synchronisation de processus (encore)

Dans le TP 5 Tubes (suite), vous avez mis en œuvre un mécanisme de synchronisation entre 2 processus.

Vous avez noté que ce mécanisme est bloquant et qu'il est possible de jouer avec l'option **`O_NONBLOCK`** pour éviter ces blocages. Si vous n'avez pas utilisé cette option, vous avez raté une étape !

Nous allons voir maintenant une autre solution pour composer avec cette contrainte du blocage : les signaux.

La problématique

2 processus (un père et un fils) doivent synchroniser leurs actions :

- Le père affiche les secondes paires à partir de **`0`**
- Le fils affiche les secondes impaires à partir de **`1`**
- Chacun fait ceci (on commence par le père) :
    - Affiche son compteur de secondes
    - Incrémente son compteur de secondes de 2 unités
    - Pause 1 seconde
    - Indique à l'autre que c'est à son tour
    - Attend que l'autre indique que c'est à notre tour
    - Boucle

Sans signal

Réaliser ce mécanisme **sans signal**. On utilise la particularité des tubes qui est que lectures et écritures sont bloquantes par défaut. Indice : ne soyez pas avare en tubes.

Avec signal

On souhaite que père et fils puissent travailler pendant leur attente au lieu d'être bloqués. Il faut donc mettre en place un système de signalisation et exécuter le code suivant (qui simule un travail quelconque) au lieu de rester bloqué en lecture de tube.

La signalisation consiste au fils à envoyer un signal vers le père pour terminer son travail et qu'il se mette en lecture du tube. Idem pour le père vers son fils. Indice : il y a quelque chose à faire avec la variable globale définie dans le code ci-dessous. Et utilisez **SIGUSR1**, par exemple, pour le signal.

Pour le père, le travail sera d'exécuter **do_job_pere()** :

```
bool fini;    // globale
void do_job_pere() {
     double pi, res;
     long loop;

     pi = 0;
     fini = false;
     loop = 1;
     while (!fini) {
          pi = pi + 1.0/(double)pow(loop, 2);
          loop++;
     }
     res = sqrt(pi * 6);
     printf("PI = %.18f (précision : %e) en %ld itérations\n",
          res, (res - 3.14159265358979323846264338327950288419 7),
          loop);
}
```

Pour le fils, le travail sera d'exécuter **do_job_fils()** :

```
void do_job_fils() {
     double pi, res;
     long loop;

     pi = 0;
     loop = 1;
     while (!fini) {
          pi = pi + 1.0 / (double)pow(loop, 4);
          loop++;
     }
     res = pow(pi * 90, 0.25);
     printf("PI = %.18f (précision : %e) en %ld itérations\n",
          res, (res - 3.14159265358979323846264338327950288419 7),
          loop);
}
```

Pour info, ces 2 codes calculent π de 2 façons différentes. Découvrez laquelle sera la plus efficace dans le temps imparti… Bougez la souris pour observer un léger ralentissement.